Problem 2326: Spiral Matrix IV

Problem Information

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

Problem Description

You are given two integers

m

and

n

, which represent the dimensions of a matrix.

You are also given the

head

of a linked list of integers.

Generate an

m x n

matrix that contains the integers in the linked list presented in

spiral

order

(clockwise)

, starting from the

top-left

of the matrix. If there are remaining empty spaces, fill them with

-1

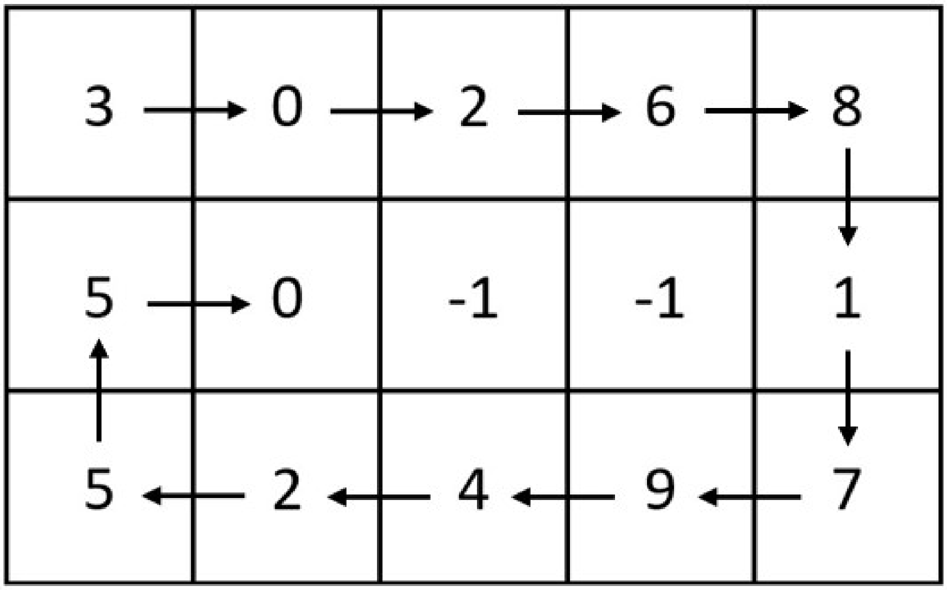
.

Return

the generated matrix

.

Example 1:



3	0	2	6	8
5	0	-1	-1	1
5	2	4	9	7

Input:

m = 3, n = 5, head = [3,0,2,6,8,1,7,9,4,2,5,5,0]

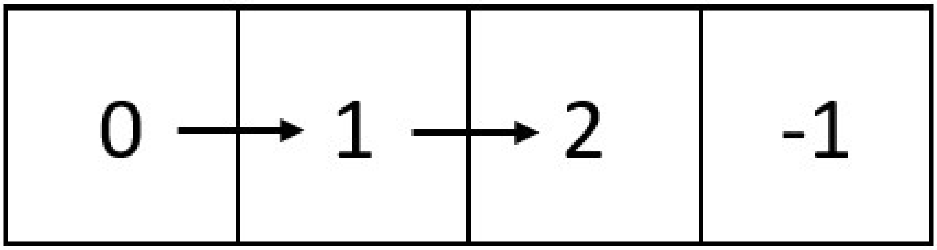
Output:

[[3,0,2,6,8],[5,0,-1,-1,1],[5,2,4,9,7]]

Explanation:

The diagram above shows how the values are printed in the matrix. Note that the remaining spaces in the matrix are filled with -1.

Example 2:



Input:

m = 1, n = 4, head = [0,1,2]

Output:

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

Explanation:

The diagram above shows how the values are printed from left to right in the matrix. The last space in the matrix is set to -1.

Constraints:

1 <= m, n <= 10

5

1 <= m * n <= 10

5

The number of nodes in the list is in the range

[1, m * n]

.

0 <= Node.val <= 1000

Code Snippets

C++:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* ListNode *next;
* ListNode() : val(0), next(nullptr) {}
* ListNode(int x) : val(x), next(nullptr) {}
* ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/
class Solution {
public:
vector<vector<int>> spiralMatrix(int m, int n, ListNode* head) {

}
};
```

Java:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
```

```
class Solution {
public int[][] spiralMatrix(int m, int n, ListNode head) {

}
}
```

Python3:

```
# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def spiralMatrix(self, m: int, n: int, head: Optional[ListNode]) ->
List[List[int]]:
```

Python:

```
# Definition for singly-linked list.
# class ListNode(object):
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution(object):
def spiralMatrix(self, m, n, head):
"""
:type m: int
:type n: int
:type head: Optional[ListNode]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* Definition for singly-linked list.
* function ListNode(val, next) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
*/
```

```
/**
* @param {number} m
* @param {number} n
* @param {ListNode} head
* @return {number[][]}
*/
var spiralMatrix = function(m, n, head) {

};
```

TypeScript:

```
/**
* Definition for singly-linked list.
* class ListNode {
* val: number
* next: ListNode | null
* constructor(val?: number, next?: ListNode | null) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
* }
*/

function spiralMatrix(m: number, n: number, head: ListNode | null):
number[][] {

};
```

C#:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public int val;
* public ListNode next;
* public ListNode(int val=0, ListNode next=null) {
* this.val = val;
* this.next = next;
* }
* }
*/
```

```
public class Solution {
public int[][] SpiralMatrix(int m, int n, ListNode head) {

}
}
```

C:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
/**
* 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** spiralMatrix(int m, int n, struct ListNode* head, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func spiralMatrix(m int, n int, head *ListNode) [][]int {

}
```

Kotlin:

```
/**
* Example:
* var li = ListNode(5)
* var v = li.`val`
* Definition for singly-linked list.
* class ListNode(var `val`: Int) {
* var next: ListNode? = null
* }
*/
class Solution {
fun spiralMatrix(m: Int, n: Int, head: ListNode?): Array<IntArray> {

}
}
```

Swift:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public var val: Int
* public var next: ListNode?
* public init() { self.val = 0; self.next = nil; }
* public init(_ val: Int) { self.val = val; self.next = nil; }
* public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next = next; }
* }
*/
class Solution {
func spiralMatrix(_ m: Int, _ n: Int, _ head: ListNode?) -> [[Int]] {

}
}
```

Rust:

```
// Definition for singly-linked list.
// #[derive(PartialEq, Eq, Clone, Debug)]
// pub struct ListNode {
// pub val: i32,
// pub next: Option<Box<ListNode>>
// }
//
```

```
// impl ListNode {
// #[inline]
// fn new(val: i32) -> Self {
// ListNode {
// next: None,
// val
// }
// }
// }
impl Solution {
pub fn spiral_matrix(m: i32, n: i32, head: Option<Box<ListNode>>) ->
Vec<Vec<i32>> {

}
}
```

Ruby:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
# @param {Integer} m
# @param {Integer} n
# @param {ListNode} head
# @return {Integer[][]}
def spiral_matrix(m, n, head)

end
```

PHP:

```
/**
* Definition for a singly-linked list.
* class ListNode {
* public $val = 0;
* public $next = null;
* function __construct($val = 0, $next = null) {
```

```
* $this->val = $val;
* $this->next = $next;
* }
* }
*/
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param ListNode $head
* @return Integer[][]
*/
function spiralMatrix($m, $n, $head) {

}
}
```

Dart:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
List<List<int>> spiralMatrix(int m, int n, ListNode? head) {

}
}
```

Scala:

```
/**
* Definition for singly-linked list.
* class ListNode(_x: Int = 0, _next: ListNode = null) {
* var next: ListNode = _next
* var x: Int = _x
* }
```

```
*/
object Solution {
def spiralMatrix(m: Int, n: Int, head: ListNode): Array[Array[Int]] = {

}
}
```

Elixir:

```
# Definition for singly-linked list.
#
# defmodule ListNode do
# @type t :: %__MODULE__{
# val: integer,
# next: ListNode.t() | nil
# }
# defstruct val: 0, next: nil
# end

defmodule Solution do
@spec spiral_matrix(m :: integer, n :: integer, head :: ListNode.t | nil) ::
[[integer]]
def spiral_matrix(m, n, head) do

end
end
```

Erlang:

```
%% Definition for singly-linked list.
%%
%% -record(list_node, {val = 0 :: integer(),
%% next = null :: 'null' | #list_node{}}).

-spec spiral_matrix(M :: integer(), N :: integer(), Head :: #list_node{} |
null) -> [[integer()]].
spiral_matrix(M, N, Head) ->
.
```

Racket:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
(val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
(list-node val #f))

|#

(define/contract (spiral-matrix m n head)
(-> exact-integer? exact-integer? (or/c list-node? #f) (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Spiral Matrix IV
* Difficulty: Medium
* Tags: array, linked_list
*
* 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
*/

/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* ListNode *next;
* ListNode() : val(0), next(nullptr) {}
* ListNode(int x) : val(x), next(nullptr) {}
* ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/
class Solution {
public:
vector<vector<int>> spiralMatrix(int m, int n, ListNode* head) {

}
};
```

Java Solution:

```
/**
* Problem: Spiral Matrix IV
* Difficulty: Medium
* Tags: array, linked_list
*
* 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
*/

/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public int[][] spiralMatrix(int m, int n, ListNode head) {

}
}
```

Python3 Solution:

```
"""
Problem: Spiral Matrix IV
```

```
Difficulty: Medium
Tags: array, linked_list

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
"""

# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def spiralMatrix(self, m: int, n: int, head: Optional[ListNode]) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for singly-linked list.
# class ListNode(object):
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution(object):
def spiralMatrix(self, m, n, head):
"""
:type m: int
:type n: int
:type head: Optional[ListNode]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Spiral Matrix IV
* Difficulty: Medium
* Tags: array, linked_list
```

```
*
* 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
*/

/**
* Definition for singly-linked list.
* function ListNode(val, next) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
*/
/**
* @param {number} m
* @param {number} n
* @param {ListNode} head
* @return {number[][]}
*/
var spiralMatrix = function(m, n, head) {

};
```

TypeScript Solution:

```
/**
* Problem: Spiral Matrix IV
* Difficulty: Medium
* Tags: array, linked_list
*
* 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
*/

/**
* Definition for singly-linked list.
* class ListNode {
* val: number
* next: ListNode | null
* constructor(val?: number, next?: ListNode | null) {
```

```
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
* }
*/

function spiralMatrix(m: number, n: number, head: ListNode | null):
number[][] {

};
```

C# Solution:

```
/*
* Problem: Spiral Matrix IV
* Difficulty: Medium
* Tags: array, linked_list
*
* 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
*/

/**
* Definition for singly-linked list.
* public class ListNode {
* public int val;
* public ListNode next;
* public ListNode(int val=0, ListNode next=null) {
* this.val = val;
* this.next = next;
* }
* }
*/
public class Solution {
public int[][] SpiralMatrix(int m, int n, ListNode head) {

}
}
```

C Solution:

```c
/*
* Problem: Spiral Matrix IV
* Difficulty: Medium
* Tags: array, linked_list
*
* 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
*/

/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
/**
* 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** spiralMatrix(int m, int n, struct ListNode* head, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```go
// Problem: Spiral Matrix IV
// Difficulty: Medium
// Tags: array, linked_list
//
// 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

/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func spiralMatrix(m int, n int, head *ListNode) [][]int {

}
```

Kotlin Solution:

```
/**
* Example:
* var li = ListNode(5)
* var v = li.`val`
* Definition for singly-linked list.
* class ListNode(var `val`: Int) {
* var next: ListNode? = null
* }
*/
class Solution {
fun spiralMatrix(m: Int, n: Int, head: ListNode?): Array<IntArray> {

}
}
```

Swift Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public var val: Int
* public var next: ListNode?
* public init() { self.val = 0; self.next = nil; }
* public init(_ val: Int) { self.val = val; self.next = nil; }
* public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next =
next; }
* }
*/
class Solution {
func spiralMatrix(_ m: Int, _ n: Int, _ head: ListNode?) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Spiral Matrix IV
// Difficulty: Medium
// Tags: array, linked_list
//
// 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

// Definition for singly-linked list.
// #[derive(PartialEq, Eq, Clone, Debug)]
// pub struct ListNode {
// pub val: i32,
// pub next: Option<Box<ListNode>>
// }
//
// impl ListNode {
// #[inline]
// fn new(val: i32) -> Self {
// ListNode {
// next: None,
// val
// }
// }
// }
impl Solution {
pub fn spiral_matrix(m: i32, n: i32, head: Option<Box<ListNode>>) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
# @param {Integer} m
# @param {Integer} n
# @param {ListNode} head
# @return {Integer[][]}
def spiral_matrix(m, n, head)

end
```

PHP Solution:

```
/**
* Definition for a singly-linked list.
* class ListNode {
* public $val = 0;
* public $next = null;
* function __construct($val = 0, $next = null) {
* $this->val = $val;
* $this->next = $next;
* }
* }
*/
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param ListNode $head
* @return Integer[][]
*/
function spiralMatrix($m, $n, $head) {

}
}
```

Dart Solution:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
List<List<int>> spiralMatrix(int m, int n, ListNode? head) {

}
}
```

Scala Solution:

```
/**
* Definition for singly-linked list.
* class ListNode(_x: Int = 0, _next: ListNode = null) {
* var next: ListNode = _next
* var x: Int = _x
* }
*/
object Solution {
def spiralMatrix(m: Int, n: Int, head: ListNode): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
# Definition for singly-linked list.
#
# defmodule ListNode do
# @type t :: %__MODULE__{
# val: integer,
# next: ListNode.t() | nil
# }
# defstruct val: 0, next: nil
# end

defmodule Solution do
```

```
@spec spiral_matrix(m :: integer, n :: integer, head :: ListNode.t | nil) ::
[[integer]]
def spiral_matrix(m, n, head) do

end
end
```

Erlang Solution:

```
%% Definition for singly-linked list.
%%
%% -record(list_node, {val = 0 :: integer(),
%% next = null :: 'null' | #list_node{}}).

-spec spiral_matrix(M :: integer(), N :: integer(), Head :: #list_node{} |
null) -> [[integer()]].
spiral_matrix(M, N, Head) ->
.
```

Racket Solution:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
(val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
(list-node val #f))

|#

(define/contract (spiral-matrix m n head)
(-> exact-integer? exact-integer? (or/c list-node? #f) (listof (listof
exact-integer?)))
)
```