# Problem 2286: Booking Concert Tickets in Groups

## Problem Information

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

## Problem Description

A concert hall has

n

rows numbered from

0

to

n - 1

, each with

m

seats, numbered from

0

to

m - 1

. You need to design a ticketing system that can allocate seats in the following cases:

If a group of

k

spectators can sit

together

in a row.

If

every

member of a group of

k

spectators can get a seat. They may or

may not

sit together.

Note that the spectators are very picky. Hence:

They will book seats only if each member of their group can get a seat with row number

less than or equal

to

maxRow

.

maxRow

can

vary

from group to group.

In case there are multiple rows to choose from, the row with the

smallest

number is chosen. If there are multiple seats to choose in the same row, the seat with the

smallest

number is chosen.

Implement the

BookMyShow

class:

BookMyShow(int n, int m)

Initializes the object with

n

as number of rows and

m

as number of seats per row.

int[] gather(int k, int maxRow)

Returns an array of length

2

denoting the row and seat number (respectively) of the

first seat

being allocated to the

k

members of the group, who must sit

together

. In other words, it returns the smallest possible

r

and

c

such that all

[c, c + k - 1]

seats are valid and empty in row

r

, and

r <= maxRow

. Returns

[]

in case it is

not possible

to allocate seats to the group.

boolean scatter(int k, int maxRow)

Returns

true

if all

k

members of the group can be allocated seats in rows

0

to

maxRow

, who may or

may not

sit together. If the seats can be allocated, it allocates

k

seats to the group with the

smallest

row numbers, and the smallest possible seat numbers in each row. Otherwise, returns

false

.

Example 1:

Input

["BookMyShow", "gather", "gather", "scatter", "scatter"] [[2, 5], [4, 0], [2, 0], [5, 1], [5, 1]]

Output

[null, [0, 0], [], true, false]

Explanation

BookMyShow bms = new BookMyShow(2, 5); // There are 2 rows with 5 seats each bms.gather(4, 0); // return [0, 0] // The group books seats [0, 3] of row 0. bms.gather(2, 0); // return [] // There is only 1 seat left in row 0, // so it is not possible to book 2 consecutive seats. bms.scatter(5, 1); // return True // The group books seat 4 of row 0 and seats [0, 3] of row 1. bms.scatter(5, 1); // return False // There is only one seat left in the hall.

Constraints:

1 <= n <= 5 * 10

4

1 <= m, k <= 10

9

0 <= maxRow <= n - 1

At most

5 * 10

4

calls

in total

will be made to

gather

and

scatter

.

## Code Snippets

**C++:**

```
class BookMyShow {
public:
BookMyShow(int n, int m) {

}

vector<int> gather(int k, int maxRow) {

}

bool scatter(int k, int maxRow) {

}
};

/**
* Your BookMyShow object will be instantiated and called as such:
* BookMyShow* obj = new BookMyShow(n, m);
* vector<int> param_1 = obj->gather(k,maxRow);
* bool param_2 = obj->scatter(k,maxRow);
*/
```

**Java:**

```
class BookMyShow {
```

```
public BookMyShow(int n, int m) {

}

public int[] gather(int k, int maxRow) {

}

public boolean scatter(int k, int maxRow) {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* BookMyShow obj = new BookMyShow(n, m);
* int[] param_1 = obj.gather(k,maxRow);
* boolean param_2 = obj.scatter(k,maxRow);
*/
```

**Python3:**

```
class BookMyShow:

def __init__(self, n: int, m: int):


def gather(self, k: int, maxRow: int) -> List[int]:


def scatter(self, k: int, maxRow: int) -> bool:



# Your BookMyShow object will be instantiated and called as such:
# obj = BookMyShow(n, m)
# param_1 = obj.gather(k,maxRow)
# param_2 = obj.scatter(k,maxRow)
```

**Python:**

```
class BookMyShow(object):

def __init__(self, n, m):
"""
:type n: int
:type m: int
"""


def gather(self, k, maxRow):
"""
:type k: int
:type maxRow: int
:rtype: List[int]
"""


def scatter(self, k, maxRow):
"""
:type k: int
:type maxRow: int
:rtype: bool
"""



# Your BookMyShow object will be instantiated and called as such:
# obj = BookMyShow(n, m)
# param_1 = obj.gather(k,maxRow)
# param_2 = obj.scatter(k,maxRow)
```

**JavaScript:**

```
/**
* @param {number} n
* @param {number} m
*/
var BookMyShow = function(n, m) {

};

/**
* @param {number} k
```

```
* @param {number} maxRow
* @return {number[]}
*/
BookMyShow.prototype.gather = function(k, maxRow) {

};

/**
* @param {number} k
* @param {number} maxRow
* @return {boolean}
*/
BookMyShow.prototype.scatter = function(k, maxRow) {

};

/**
* Your BookMyShow object will be instantiated and called as such:
* var obj = new BookMyShow(n, m)
* var param_1 = obj.gather(k,maxRow)
* var param_2 = obj.scatter(k,maxRow)
*/
```

**TypeScript:**

```
class BookMyShow {
constructor(n: number, m: number) {

}

gather(k: number, maxRow: number): number[] {

}

scatter(k: number, maxRow: number): boolean {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* var obj = new BookMyShow(n, m)
```

```
* var param_1 = obj.gather(k,maxRow)
* var param_2 = obj.scatter(k,maxRow)
*/
```

**C#:**

```
public class BookMyShow {

public BookMyShow(int n, int m) {

}

public int[] Gather(int k, int maxRow) {

}

public bool Scatter(int k, int maxRow) {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* BookMyShow obj = new BookMyShow(n, m);
* int[] param_1 = obj.Gather(k,maxRow);
* bool param_2 = obj.Scatter(k,maxRow);
*/
```

**C:**

```
typedef struct {

} BookMyShow;


BookMyShow* bookMyShowCreate(int n, int m) {

}
```

```
int* bookMyShowGather(BookMyShow* obj, int k, int maxRow, int* retSize) {

}

bool bookMyShowScatter(BookMyShow* obj, int k, int maxRow) {

}

void bookMyShowFree(BookMyShow* obj) {

}

/**
* Your BookMyShow struct will be instantiated and called as such:
* BookMyShow* obj = bookMyShowCreate(n, m);
* int* param_1 = bookMyShowGather(obj, k, maxRow, retSize);

* bool param_2 = bookMyShowScatter(obj, k, maxRow);

* bookMyShowFree(obj);
*/
```

**Go:**

```
type BookMyShow struct {

}


func Constructor(n int, m int) BookMyShow {

}


func (this *BookMyShow) Gather(k int, maxRow int) []int {

}


func (this *BookMyShow) Scatter(k int, maxRow int) bool {

}
```

```
/**
* Your BookMyShow object will be instantiated and called as such:
* obj := Constructor(n, m);
* param_1 := obj.Gather(k,maxRow);
* param_2 := obj.Scatter(k,maxRow);
*/
```

**Kotlin:**

```
class BookMyShow(n: Int, m: Int) {

fun gather(k: Int, maxRow: Int): IntArray {

}

fun scatter(k: Int, maxRow: Int): Boolean {

}

}

/**
* Your BookMyShow object will be instantiated and called as such:
* var obj = BookMyShow(n, m)
* var param_1 = obj.gather(k,maxRow)
* var param_2 = obj.scatter(k,maxRow)
*/
```

**Swift:**

```
class BookMyShow {

init(_ n: Int, _ m: Int) {

}

func gather(_ k: Int, _ maxRow: Int) -> [Int] {

}
```

```
func scatter(_ k: Int, _ maxRow: Int) -> Bool {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* let obj = BookMyShow(n, m)
* let ret_1: [Int] = obj.gather(k, maxRow)
* let ret_2: Bool = obj.scatter(k, maxRow)
*/
```

**Rust:**

```
struct BookMyShow {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl BookMyShow {

fn new(n: i32, m: i32) -> Self {

}

fn gather(&self, k: i32, max_row: i32) -> Vec<i32> {

}

fn scatter(&self, k: i32, max_row: i32) -> bool {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* let obj = BookMyShow::new(n, m);
```

```
* let ret_1: Vec<i32> = obj.gather(k, maxRow);
* let ret_2: bool = obj.scatter(k, maxRow);
*/
```

**Ruby:**

```
class BookMyShow

=begin
:type n: Integer
:type m: Integer
=end
def initialize(n, m)

end


=begin
:type k: Integer
:type max_row: Integer
:rtype: Integer[]
=end
def gather(k, max_row)

end


=begin
:type k: Integer
:type max_row: Integer
:rtype: Boolean
=end
def scatter(k, max_row)

end


end

# Your BookMyShow object will be instantiated and called as such:
# obj = BookMyShow.new(n, m)
# param_1 = obj.gather(k, max_row)
```

```
# param_2 = obj.scatter(k, max_row)
```

**PHP:**

```
class BookMyShow {
/**
* @param Integer $n
* @param Integer $m
*/
function __construct($n, $m) {

}

/**
* @param Integer $k
* @param Integer $maxRow
* @return Integer[]
*/
function gather($k, $maxRow) {

}

/**
* @param Integer $k
* @param Integer $maxRow
* @return Boolean
*/
function scatter($k, $maxRow) {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* $obj = BookMyShow($n, $m);
* $ret_1 = $obj->gather($k, $maxRow);
* $ret_2 = $obj->scatter($k, $maxRow);
*/
```

**Dart:**

```
class BookMyShow {

BookMyShow(int n, int m) {

}

List<int> gather(int k, int maxRow) {

}

bool scatter(int k, int maxRow) {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* BookMyShow obj = BookMyShow(n, m);
* List<int> param1 = obj.gather(k,maxRow);
* bool param2 = obj.scatter(k,maxRow);
*/
```

**Scala:**

```
class BookMyShow(_n: Int, _m: Int) {

def gather(k: Int, maxRow: Int): Array[Int] = {

}

def scatter(k: Int, maxRow: Int): Boolean = {

}

}

/**
* Your BookMyShow object will be instantiated and called as such:
* val obj = new BookMyShow(n, m)
* val param_1 = obj.gather(k,maxRow)
* val param_2 = obj.scatter(k,maxRow)
*/
```

**Elixir:**

```
defmodule BookMyShow do
@spec init_(n :: integer, m :: integer) :: any
def init_(n, m) do

end

@spec gather(k :: integer, max_row :: integer) :: [integer]
def gather(k, max_row) do

end

@spec scatter(k :: integer, max_row :: integer) :: boolean
def scatter(k, max_row) do

end
end

# Your functions will be called as such:
# BookMyShow.init_(n, m)
# param_1 = BookMyShow.gather(k, max_row)
# param_2 = BookMyShow.scatter(k, max_row)

# BookMyShow.init_ will be called before every test case, in which you can do
some necessary initializations.
```

**Erlang:**

```
-spec book_my_show_init_(N :: integer(), M :: integer()) -> any().
book_my_show_init_(N, M) ->
.

-spec book_my_show_gather(K :: integer(), MaxRow :: integer()) ->
[integer()].
book_my_show_gather(K, MaxRow) ->
.

-spec book_my_show_scatter(K :: integer(), MaxRow :: integer()) -> boolean().
book_my_show_scatter(K, MaxRow) ->
.
```

```
%% Your functions will be called as such:
%% book_my_show_init_(N, M),
%% Param_1 = book_my_show_gather(K, MaxRow),
%% Param_2 = book_my_show_scatter(K, MaxRow),

%% book_my_show_init_ will be called before every test case, in which you can
do some necessary initializations.
```

**Racket:**

```
(define book-my-show%
(class object%
(super-new)

; n : exact-integer?
; m : exact-integer?
(init-field
n
m)

; gather : exact-integer? exact-integer? -> (listof exact-integer?)
(define/public (gather k max-row)
)
; scatter : exact-integer? exact-integer? -> boolean?
(define/public (scatter k max-row)
)))

;; Your book-my-show% object will be instantiated and called as such:
;; (define obj (new book-my-show% [n n] [m m]))
;; (define param_1 (send obj gather k max-row))
;; (define param_2 (send obj scatter k max-row))
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Booking Concert Tickets in Groups
* Difficulty: Hard
```

```
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class BookMyShow {
public:
BookMyShow(int n, int m) {

}

vector<int> gather(int k, int maxRow) {

}

bool scatter(int k, int maxRow) {

}
};

/**
* Your BookMyShow object will be instantiated and called as such:
* BookMyShow* obj = new BookMyShow(n, m);
* vector<int> param_1 = obj->gather(k,maxRow);
* bool param_2 = obj->scatter(k,maxRow);
*/
```

**Java Solution:**

```
/**
* Problem: Booking Concert Tickets in Groups
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
class BookMyShow {

public BookMyShow(int n, int m) {

}

public int[] gather(int k, int maxRow) {

}

public boolean scatter(int k, int maxRow) {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* BookMyShow obj = new BookMyShow(n, m);
* int[] param_1 = obj.gather(k,maxRow);
* boolean param_2 = obj.scatter(k,maxRow);
*/
```

**Python3 Solution:**

```
"""
Problem: Booking Concert Tickets in Groups
Difficulty: Hard
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class BookMyShow:

def __init__(self, n: int, m: int):


def gather(self, k: int, maxRow: int) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

### Python Solution:

```
class BookMyShow(object):

def __init__(self, n, m):
"""
:type n: int
:type m: int
"""



def gather(self, k, maxRow):
"""
:type k: int
:type maxRow: int
:rtype: List[int]
"""



def scatter(self, k, maxRow):
"""
:type k: int
:type maxRow: int
:rtype: bool
"""




# Your BookMyShow object will be instantiated and called as such:
# obj = BookMyShow(n, m)
# param_1 = obj.gather(k,maxRow)
# param_2 = obj.scatter(k,maxRow)
```

### JavaScript Solution:

```
/**
* Problem: Booking Concert Tickets in Groups
* Difficulty: Hard
* Tags: array, tree, search
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number} n
* @param {number} m
*/
var BookMyShow = function(n, m) {

};

/**
* @param {number} k
* @param {number} maxRow
* @return {number[]}
*/
BookMyShow.prototype.gather = function(k, maxRow) {

};

/**
* @param {number} k
* @param {number} maxRow
* @return {boolean}
*/
BookMyShow.prototype.scatter = function(k, maxRow) {

};

/**
* Your BookMyShow object will be instantiated and called as such:
* var obj = new BookMyShow(n, m)
* var param_1 = obj.gather(k,maxRow)
* var param_2 = obj.scatter(k,maxRow)
*/
```

**TypeScript Solution:**

```
/**
* Problem: Booking Concert Tickets in Groups
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class BookMyShow {
constructor(n: number, m: number) {

}

gather(k: number, maxRow: number): number[] {

}

scatter(k: number, maxRow: number): boolean {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* var obj = new BookMyShow(n, m)
* var param_1 = obj.gather(k,maxRow)
* var param_2 = obj.scatter(k,maxRow)
*/
```

**C# Solution:**

```
/*
* Problem: Booking Concert Tickets in Groups
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
public class BookMyShow {

public BookMyShow(int n, int m) {

}

public int[] Gather(int k, int maxRow) {

}

public bool Scatter(int k, int maxRow) {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* BookMyShow obj = new BookMyShow(n, m);
* int[] param_1 = obj.Gather(k,maxRow);
* bool param_2 = obj.Scatter(k,maxRow);
*/
```

### C Solution:

```
/*
* Problem: Booking Concert Tickets in Groups
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/




typedef struct {

} BookMyShow;
```

```
BookMyShow* bookMyShowCreate(int n, int m) {

}

int* bookMyShowGather(BookMyShow* obj, int k, int maxRow, int* retSize) {

}

bool bookMyShowScatter(BookMyShow* obj, int k, int maxRow) {

}

void bookMyShowFree(BookMyShow* obj) {

}

/**
* Your BookMyShow struct will be instantiated and called as such:
* BookMyShow* obj = bookMyShowCreate(n, m);
* int* param_1 = bookMyShowGather(obj, k, maxRow, retSize);

* bool param_2 = bookMyShowScatter(obj, k, maxRow);

* bookMyShowFree(obj);
*/
```

### Go Solution:

```
// Problem: Booking Concert Tickets in Groups
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

type BookMyShow struct {

}
```

```
func Constructor(n int, m int) BookMyShow {

}


func (this *BookMyShow) Gather(k int, maxRow int) []int {

}


func (this *BookMyShow) Scatter(k int, maxRow int) bool {

}


/**
* Your BookMyShow object will be instantiated and called as such:
* obj := Constructor(n, m);
* param_1 := obj.Gather(k,maxRow);
* param_2 := obj.Scatter(k,maxRow);
*/
```

**Kotlin Solution:**

```
class BookMyShow(n: Int, m: Int) {

fun gather(k: Int, maxRow: Int): IntArray {

}

fun scatter(k: Int, maxRow: Int): Boolean {

}

}

/**
* Your BookMyShow object will be instantiated and called as such:
* var obj = BookMyShow(n, m)
```

```
* var param_1 = obj.gather(k,maxRow)
* var param_2 = obj.scatter(k,maxRow)
*/
```

**Swift Solution:**

```
class BookMyShow {

init(_ n: Int, _ m: Int) {

}

func gather(_ k: Int, _ maxRow: Int) -> [Int] {

}

func scatter(_ k: Int, _ maxRow: Int) -> Bool {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* let obj = BookMyShow(n, m)
* let ret_1: [Int] = obj.gather(k, maxRow)
* let ret_2: Bool = obj.scatter(k, maxRow)
*/
```

**Rust Solution:**

```
// Problem: Booking Concert Tickets in Groups
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

struct BookMyShow {
```

```
}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl BookMyShow {

fn new(n: i32, m: i32) -> Self {

}

fn gather(&self, k: i32, max_row: i32) -> Vec<i32> {

}

fn scatter(&self, k: i32, max_row: i32) -> bool {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* let obj = BookMyShow::new(n, m);
* let ret_1: Vec<i32> = obj.gather(k, maxRow);
* let ret_2: bool = obj.scatter(k, maxRow);
*/
```

**Ruby Solution:**

```
class BookMyShow

=begin
:type n: Integer
:type m: Integer
=end
def initialize(n, m)

end
```

```
=begin
:type k: Integer
:type max_row: Integer
:rtype: Integer[]
=end
def gather(k, max_row)

end


=begin
:type k: Integer
:type max_row: Integer
:rtype: Boolean
=end
def scatter(k, max_row)

end



end

# Your BookMyShow object will be instantiated and called as such:
# obj = BookMyShow.new(n, m)
# param_1 = obj.gather(k, max_row)
# param_2 = obj.scatter(k, max_row)
```

**PHP Solution:**

```
class BookMyShow {
/**
* @param Integer $n
* @param Integer $m
*/
function __construct($n, $m) {

}

/**
* @param Integer $k
```

```
* @param Integer $maxRow
* @return Integer[]
*/
function gather($k, $maxRow) {

}

/**
* @param Integer $k
* @param Integer $maxRow
* @return Boolean
*/
function scatter($k, $maxRow) {

}
}

/**
* Your BookMyShow object will be instantiated and called as such:
* $obj = BookMyShow($n, $m);
* $ret_1 = $obj->gather($k, $maxRow);
* $ret_2 = $obj->scatter($k, $maxRow);
*/
```

### Dart Solution:

```
class BookMyShow {

BookMyShow(int n, int m) {

}

List<int> gather(int k, int maxRow) {

}

bool scatter(int k, int maxRow) {

}
}
```

```
/**
* Your BookMyShow object will be instantiated and called as such:
* BookMyShow obj = BookMyShow(n, m);
* List<int> param1 = obj.gather(k,maxRow);
* bool param2 = obj.scatter(k,maxRow);
*/
```

### Scala Solution:

```
class BookMyShow(_n: Int, _m: Int) {

def gather(k: Int, maxRow: Int): Array[Int] = {

}

def scatter(k: Int, maxRow: Int): Boolean = {

}

}

/**
* Your BookMyShow object will be instantiated and called as such:
* val obj = new BookMyShow(n, m)
* val param_1 = obj.gather(k,maxRow)
* val param_2 = obj.scatter(k,maxRow)
*/
```

### Elixir Solution:

```
defmodule BookMyShow do
@spec init_(n :: integer, m :: integer) :: any
def init_(n, m) do

end

@spec gather(k :: integer, max_row :: integer) :: [integer]
def gather(k, max_row) do

end
```

```
@spec scatter(k :: integer, max_row :: integer) :: boolean
def scatter(k, max_row) do

end
end

# Your functions will be called as such:
# BookMyShow.init_(n, m)
# param_1 = BookMyShow.gather(k, max_row)
# param_2 = BookMyShow.scatter(k, max_row)

# BookMyShow.init_ will be called before every test case, in which you can do some necessary initializations.
```

**Erlang Solution:**

```
-spec book_my_show_init_(N :: integer(), M :: integer()) -> any().
book_my_show_init_(N, M) ->
.

-spec book_my_show_gather(K :: integer(), MaxRow :: integer()) ->
[integer()].
book_my_show_gather(K, MaxRow) ->
.

-spec book_my_show_scatter(K :: integer(), MaxRow :: integer()) -> boolean().
book_my_show_scatter(K, MaxRow) ->
.


%% Your functions will be called as such:
%% book_my_show_init_(N, M),
%% Param_1 = book_my_show_gather(K, MaxRow),
%% Param_2 = book_my_show_scatter(K, MaxRow),

%% book_my_show_init_ will be called before every test case, in which you can do some necessary initializations.
```

**Racket Solution:**

```
(define book-my-show%
(class object%
(super-new)

; n : exact-integer?
; m : exact-integer?
(init-field
n
m)

; gather : exact-integer? exact-integer? -> (listof exact-integer?)
(define/public (gather k max-row)
)
; scatter : exact-integer? exact-integer? -> boolean?
(define/public (scatter k max-row)
)))

;; Your book-my-show% object will be instantiated and called as such:
;; (define obj (new book-my-show% [n n] [m m]))
;; (define param_1 (send obj gather k max-row))
;; (define param_2 (send obj scatter k max-row))
```