Problem 1912: Design Movie Rental System

Problem Information

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

Problem Description

You have a movie renting company consisting of

n

shops. You want to implement a renting system that supports searching for, booking, and returning movies. The system should also support generating a report of the currently rented movies.

Each movie is given as a 2D integer array

entries

where

entries[i] = [shop

i

, movie

i

, price

i

]

indicates that there is a copy of movie

movie

i

at shop

shop

i

with a rental price of

price

i

. Each shop carries

at most one

copy of a movie

movie

i

.

The system should support the following functions:

Search

: Finds the

cheapest 5 shops

that have an

unrented copy

of a given movie. The shops should be sorted by

price

in ascending order, and in case of a tie, the one with the

smaller

shop

i

should appear first. If there are less than 5 matching shops, then all of them should be returned. If no shop has an unrented copy, then an empty list should be returned.

Rent

: Rents an

unrented copy

of a given movie from a given shop.

Drop

: Drops off a

previously rented copy

of a given movie at a given shop.

Report

: Returns the

cheapest 5 rented movies

(possibly of the same movie ID) as a 2D list

res

where

res[j] = [shop

j

, movie

j

]

describes that the

j

th

cheapest rented movie

movie

j

was rented from the shop

shop

j

. The movies in

res

should be sorted by

price

in ascending order, and in case of a tie, the one with the

smaller

shop

j

should appear first, and if there is still tie, the one with the

smaller

movie

j

should appear first. If there are fewer than 5 rented movies, then all of them should be returned. If no movies are currently being rented, then an empty list should be returned.

Implement the

MovieRentingSystem

class:

MovieRentingSystem(int n, int[][] entries)

Initializes the

MovieRentingSystem

object with

n

shops and the movies in

entries

.

List<Integer> search(int movie)

Returns a list of shops that have an

unrented copy

of the given

movie

as described above.

void rent(int shop, int movie)

Rents the given

movie

from the given

shop

.

void drop(int shop, int movie)

Drops off a previously rented

movie

at the given

shop

.

List<List<Integer>> report()

Returns a list of cheapest

rented

movies as described above.

Note:

The test cases will be generated such that

rent

will only be called if the shop has an

unrented

copy of the movie, and

drop

will only be called if the shop had

previously rented

out the movie.

Example 1:

Input

["MovieRentingSystem", "search", "rent", "rent", "report", "drop", "search"] [[3, [[0, 1, 5], [0, 2, 6], [0, 3, 7], [1, 1, 4], [1, 2, 7], [2, 1, 5]]], [1], [0, 1], [1, 2], [], [1, 2], [2]] Output

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

Explanation

MovieRentingSystem movieRentingSystem = new MovieRentingSystem(3, [[0, 1, 5], [0, 2, 6], [0, 3, 7], [1, 1, 4], [1, 2, 7], [2, 1, 5]]); movieRentingSystem.search(1); // return [1, 0, 2], Movies of ID 1 are unrented at shops 1, 0, and 2. Shop 1 is cheapest; shop 0 and 2 are the same price, so order by shop number. movieRentingSystem.rent(0, 1); // Rent movie 1 from shop 0. Unrented movies at shop 0 are now [2,3]. movieRentingSystem.rent(1, 2); // Rent movie 2 from shop 1. Unrented movies at shop 1 are now [1]. movieRentingSystem.report(); // return [[0, 1], [1, 2]]. Movie 1 from shop 0 is cheapest, followed by movie 2 from shop 1. movieRentingSystem.drop(1, 2); // Drop off movie 2 at shop 1. Unrented movies at shop 1 are now [1,2]. movieRentingSystem.search(2); // return [0, 1]. Movies of ID 2 are unrented at shops 0 and 1. Shop 0 is cheapest, followed by shop 1.

Constraints:

1 <= n <= 3 * 10

5

1 <= entries.length <= 10

5

0 <= shop

i

< n

1 <= movie

i

, price

i

<= 10

4

Each shop carries

at most one

copy of a movie

movie

i

.

At most

10

5

calls

in total

will be made to

search

,

rent

,

drop

and

report

.

Code Snippets

C++:

```
class MovieRentingSystem {
public:
MovieRentingSystem(int n, vector<vector<int>>& entries) {

}

vector<int> search(int movie) {

}

void rent(int shop, int movie) {

}

void drop(int shop, int movie) {

}

vector<vector<int>> report() {

}
};

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* MovieRentingSystem* obj = new MovieRentingSystem(n, entries);
* vector<int> param_1 = obj->search(movie);
* obj->rent(shop,movie);
* obj->drop(shop,movie);
* vector<vector<int>> param_4 = obj->report();
```

```
*/
```

Java:

```
class MovieRentingSystem {

public MovieRentingSystem(int n, int[][] entries) {

}

public List<Integer> search(int movie) {

}

public void rent(int shop, int movie) {

}

public void drop(int shop, int movie) {

}

public List<List<Integer>> report() {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* MovieRentingSystem obj = new MovieRentingSystem(n, entries);
* List<Integer> param_1 = obj.search(movie);
* obj.rent(shop,movie);
* obj.drop(shop,movie);
* List<List<Integer>> param_4 = obj.report();
*/
```

Python3:

```
class MovieRentingSystem:

def __init__(self, n: int, entries: List[List[int]]):
```

```
def search(self, movie: int) -> List[int]:


def rent(self, shop: int, movie: int) -> None:


def drop(self, shop: int, movie: int) -> None:


def report(self) -> List[List[int]]:



# Your MovieRentingSystem object will be instantiated and called as such:
# obj = MovieRentingSystem(n, entries)
# param_1 = obj.search(movie)
# obj.rent(shop,movie)
# obj.drop(shop,movie)
# param_4 = obj.report()
```

Python:

```
class MovieRentingSystem(object):

def __init__(self, n, entries):
"""
:type n: int
:type entries: List[List[int]]
"""


def search(self, movie):
"""
:type movie: int
:rtype: List[int]
"""


def rent(self, shop, movie):
"""
:type shop: int
```

```
:type movie: int
:rtype: None
"""


def drop(self, shop, movie):
"""
:type shop: int
:type movie: int
:rtype: None
"""


def report(self):
"""
:rtype: List[List[int]]
"""



# Your MovieRentingSystem object will be instantiated and called as such:
# obj = MovieRentingSystem(n, entries)
# param_1 = obj.search(movie)
# obj.rent(shop,movie)
# obj.drop(shop,movie)
# param_4 = obj.report()
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} entries
*/
var MovieRentingSystem = function(n, entries) {

};

/**
* @param {number} movie
* @return {number[]}
*/
MovieRentingSystem.prototype.search = function(movie) {
};

/**
* @param {number} shop
* @param {number} movie
* @return {void}
*/
MovieRentingSystem.prototype.rent = function(shop, movie) {

};

/**
* @param {number} shop
* @param {number} movie
* @return {void}
*/
MovieRentingSystem.prototype.drop = function(shop, movie) {

};

/**
* @return {number[][]}
*/
MovieRentingSystem.prototype.report = function() {

};

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* var obj = new MovieRentingSystem(n, entries)
* var param_1 = obj.search(movie)
* obj.rent(shop,movie)
* obj.drop(shop,movie)
* var param_4 = obj.report()
*/
```

TypeScript:

```
class MovieRentingSystem {
constructor(n: number, entries: number[][]) {
}

search(movie: number): number[] {

}

rent(shop: number, movie: number): void {

}

drop(shop: number, movie: number): void {

}

report(): number[][] {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* var obj = new MovieRentingSystem(n, entries)
* var param_1 = obj.search(movie)
* obj.rent(shop,movie)
* obj.drop(shop,movie)
* var param_4 = obj.report()
*/
```

C#:

```
public class MovieRentingSystem {

public MovieRentingSystem(int n, int[][] entries) {

}

public IList<int> Search(int movie) {

}

public void Rent(int shop, int movie) {
}

public void Drop(int shop, int movie) {

}

public IList<IList<int>> Report() {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* MovieRentingSystem obj = new MovieRentingSystem(n, entries);
* IList<int> param_1 = obj.Search(movie);
* obj.Rent(shop,movie);
* obj.Drop(shop,movie);
* IList<IList<int>> param_4 = obj.Report();
*/
```

C:

```



typedef struct {

} MovieRentingSystem;


MovieRentingSystem* movieRentingSystemCreate(int n, int** entries, int
entriesSize, int* entriesColSize) {

}

int* movieRentingSystemSearch(MovieRentingSystem* obj, int movie, int*
retSize) {

}

void movieRentingSystemRent(MovieRentingSystem* obj, int shop, int movie) {
}

void movieRentingSystemDrop(MovieRentingSystem* obj, int shop, int movie) {

}

int** movieRentingSystemReport(MovieRentingSystem* obj, int* retSize, int**
retColSize) {

}

void movieRentingSystemFree(MovieRentingSystem* obj) {

}

/**
* Your MovieRentingSystem struct will be instantiated and called as such:
* MovieRentingSystem* obj = movieRentingSystemCreate(n, entries, entriesSize,
entriesColSize);
* int* param_1 = movieRentingSystemSearch(obj, movie, retSize);

* movieRentingSystemRent(obj, shop, movie);

* movieRentingSystemDrop(obj, shop, movie);

* int** param_4 = movieRentingSystemReport(obj, retSize, retColSize);

* movieRentingSystemFree(obj);
*/
```

Go:

```
type MovieRentingSystem struct {

}



func Constructor(n int, entries [][]int) MovieRentingSystem {

}
```

```
func (this *MovieRentingSystem) Search(movie int) []int {

}


func (this *MovieRentingSystem) Rent(shop int, movie int) {

}


func (this *MovieRentingSystem) Drop(shop int, movie int) {

}


func (this *MovieRentingSystem) Report() [][]int {

}


/**
* Your MovieRentingSystem object will be instantiated and called as such:
* obj := Constructor(n, entries);
* param_1 := obj.Search(movie);
* obj.Rent(shop,movie);
* obj.Drop(shop,movie);
* param_4 := obj.Report();
*/
```

Kotlin:

```
class MovieRentingSystem(n: Int, entries: Array<IntArray>) {

fun search(movie: Int): List<Int> {

}

fun rent(shop: Int, movie: Int) {

}

fun drop(shop: Int, movie: Int) {
}

fun report(): List<List<Int>> {

}

}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* var obj = MovieRentingSystem(n, entries)
* var param_1 = obj.search(movie)
* obj.rent(shop,movie)
* obj.drop(shop,movie)
* var param_4 = obj.report()
*/
```

Swift:

```
class MovieRentingSystem {

init(_ n: Int, _ entries: [[Int]]) {

}

func search(_ movie: Int) -> [Int] {

}

func rent(_ shop: Int, _ movie: Int) {

}

func drop(_ shop: Int, _ movie: Int) {

}

func report() -> [[Int]] {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* let obj = MovieRentingSystem(n, entries)
* let ret_1: [Int] = obj.search(movie)
* obj.rent(shop, movie)
* obj.drop(shop, movie)
* let ret_4: [[Int]] = obj.report()
*/
```

Rust:

```
struct MovieRentingSystem {

}



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

fn new(n: i32, entries: Vec<Vec<i32>>) -> Self {

}

fn search(&self, movie: i32) -> Vec<i32> {

}

fn rent(&self, shop: i32, movie: i32) {

}

fn drop(&self, shop: i32, movie: i32) {

}

fn report(&self) -> Vec<Vec<i32>> {
}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* let obj = MovieRentingSystem::new(n, entries);
* let ret_1: Vec<i32> = obj.search(movie);
* obj.rent(shop, movie);
* obj.drop(shop, movie);
* let ret_4: Vec<Vec<i32>> = obj.report();
*/
```

Ruby:

```
class MovieRentingSystem

=begin
:type n: Integer
:type entries: Integer[][]
=end
def initialize(n, entries)

end



=begin
:type movie: Integer
:rtype: Integer[]
=end
def search(movie)

end



=begin
:type shop: Integer
:type movie: Integer
:rtype: Void
=end
def rent(shop, movie)

end
```

```



=begin
:type shop: Integer
:type movie: Integer
:rtype: Void
=end
def drop(shop, movie)

end


=begin
:rtype: Integer[][]
=end
def report()

end


end

# Your MovieRentingSystem object will be instantiated and called as such:
# obj = MovieRentingSystem.new(n, entries)
# param_1 = obj.search(movie)
# obj.rent(shop, movie)
# obj.drop(shop, movie)
# param_4 = obj.report()
```

PHP:

```
class MovieRentingSystem {
/**
* @param Integer $n
* @param Integer[][] $entries
*/
function __construct($n, $entries) {

}

/**
* @param Integer $movie
* @return Integer[]
*/
function search($movie) {

}

/**
* @param Integer $shop
* @param Integer $movie
* @return NULL
*/
function rent($shop, $movie) {

}

/**
* @param Integer $shop
* @param Integer $movie
* @return NULL
*/
function drop($shop, $movie) {

}

/**
* @return Integer[][]
*/
function report() {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* $obj = MovieRentingSystem($n, $entries);
* $ret_1 = $obj->search($movie);
* $obj->rent($shop, $movie);
* $obj->drop($shop, $movie);
* $ret_4 = $obj->report();
*/
```

Dart:

```
class MovieRentingSystem {

MovieRentingSystem(int n, List<List<int>> entries) {

}

List<int> search(int movie) {

}

void rent(int shop, int movie) {

}

void drop(int shop, int movie) {

}

List<List<int>> report() {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* MovieRentingSystem obj = MovieRentingSystem(n, entries);
* List<int> param1 = obj.search(movie);
* obj.rent(shop,movie);
* obj.drop(shop,movie);
* List<List<int>> param4 = obj.report();
*/
```

Scala:

```
class MovieRentingSystem(_n: Int, _entries: Array[Array[Int]]) {

def search(movie: Int): List[Int] = {

}

def rent(shop: Int, movie: Int): Unit = {
}

def drop(shop: Int, movie: Int): Unit = {

}

def report(): List[List[Int]] = {

}

}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* val obj = new MovieRentingSystem(n, entries)
* val param_1 = obj.search(movie)
* obj.rent(shop,movie)
* obj.drop(shop,movie)
* val param_4 = obj.report()
*/
```

Elixir:

```
defmodule MovieRentingSystem do
@spec init_(n :: integer, entries :: [[integer]]) :: any
def init_(n, entries) do

end

@spec search(movie :: integer) :: [integer]
def search(movie) do

end

@spec rent(shop :: integer, movie :: integer) :: any
def rent(shop, movie) do

end

@spec drop(shop :: integer, movie :: integer) :: any
def drop(shop, movie) do
```

```


end

@spec report() :: [[integer]]
def report() do

end
end

# Your functions will be called as such:
# MovieRentingSystem.init_(n, entries)
# param_1 = MovieRentingSystem.search(movie)
# MovieRentingSystem.rent(shop, movie)
# MovieRentingSystem.drop(shop, movie)
# param_4 = MovieRentingSystem.report()

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

Erlang:

```
-spec movie_renting_system_init_(N :: integer(), Entries :: [[integer()]]) ->
any().
movie_renting_system_init_(N, Entries) ->
.

-spec movie_renting_system_search(Movie :: integer()) -> [integer()].
movie_renting_system_search(Movie) ->
.

-spec movie_renting_system_rent(Shop :: integer(), Movie :: integer()) ->
any().
movie_renting_system_rent(Shop, Movie) ->
.

-spec movie_renting_system_drop(Shop :: integer(), Movie :: integer()) ->
any().
movie_renting_system_drop(Shop, Movie) ->
.

-spec movie_renting_system_report() -> [[integer()]].
movie_renting_system_report() ->
```

```
.



%% Your functions will be called as such:
%% movie_renting_system_init_(N, Entries),
%% Param_1 = movie_renting_system_search(Movie),
%% movie_renting_system_rent(Shop, Movie),
%% movie_renting_system_drop(Shop, Movie),
%% Param_4 = movie_renting_system_report(),

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

Racket:

```
(define movie-renting-system%
(class object%
(super-new)

; n : exact-integer?
; entries : (listof (listof exact-integer?))
(init-field
n
entries)

; search : exact-integer? -> (listof exact-integer?)
(define/public (search movie)
)
; rent : exact-integer? exact-integer? -> void?
(define/public (rent shop movie)
)
; drop : exact-integer? exact-integer? -> void?
(define/public (drop shop movie)
)
; report : -> (listof (listof exact-integer?))
(define/public (report)
)))

;; Your movie-renting-system% object will be instantiated and called as such:
;; (define obj (new movie-renting-system% [n n] [entries entries]))
;; (define param_1 (send obj search movie))
;; (send obj rent shop movie)
;; (send obj drop shop movie)
;; (define param_4 (send obj report))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Movie Rental System
* Difficulty: Hard
* Tags: array, hash, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class MovieRentingSystem {
public:
MovieRentingSystem(int n, vector<vector<int>>& entries) {

}

vector<int> search(int movie) {

}

void rent(int shop, int movie) {

}

void drop(int shop, int movie) {

}

vector<vector<int>> report() {

}
};
```

```
/**
* Your MovieRentingSystem object will be instantiated and called as such:
* MovieRentingSystem* obj = new MovieRentingSystem(n, entries);
* vector<int> param_1 = obj->search(movie);
* obj->rent(shop,movie);
* obj->drop(shop,movie);
* vector<vector<int>> param_4 = obj->report();
*/
```

Java Solution:

```
/**
* Problem: Design Movie Rental System
* Difficulty: Hard
* Tags: array, hash, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class MovieRentingSystem {

public MovieRentingSystem(int n, int[][] entries) {

}

public List<Integer> search(int movie) {

}

public void rent(int shop, int movie) {

}

public void drop(int shop, int movie) {

}

public List<List<Integer>> report() {
}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* MovieRentingSystem obj = new MovieRentingSystem(n, entries);
* List<Integer> param_1 = obj.search(movie);
* obj.rent(shop,movie);
* obj.drop(shop,movie);
* List<List<Integer>> param_4 = obj.report();
*/
```

Python3 Solution:

```
"""
Problem: Design Movie Rental System
Difficulty: Hard
Tags: array, hash, sort, search, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class MovieRentingSystem:

def __init__(self, n: int, entries: List[List[int]]):



def search(self, movie: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class MovieRentingSystem(object):

def __init__(self, n, entries):
"""
:type n: int
:type entries: List[List[int]]
```

```
"""



def search(self, movie):
"""
:type movie: int
:rtype: List[int]
"""



def rent(self, shop, movie):
"""
:type shop: int
:type movie: int
:rtype: None
"""



def drop(self, shop, movie):
"""
:type shop: int
:type movie: int
:rtype: None
"""



def report(self):
"""
:rtype: List[List[int]]
"""




# Your MovieRentingSystem object will be instantiated and called as such:
# obj = MovieRentingSystem(n, entries)
# param_1 = obj.search(movie)
# obj.rent(shop,movie)
# obj.drop(shop,movie)
# param_4 = obj.report()
```

JavaScript Solution:

```
/**
* Problem: Design Movie Rental System
* Difficulty: Hard
* Tags: array, hash, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} n
* @param {number[][]} entries
*/
var MovieRentingSystem = function(n, entries) {

};

/**
* @param {number} movie
* @return {number[]}
*/
MovieRentingSystem.prototype.search = function(movie) {

};

/**
* @param {number} shop
* @param {number} movie
* @return {void}
*/
MovieRentingSystem.prototype.rent = function(shop, movie) {

};

/**
* @param {number} shop
* @param {number} movie
* @return {void}
*/
MovieRentingSystem.prototype.drop = function(shop, movie) {

};
```

```
/**
* @return {number[][]}
*/
MovieRentingSystem.prototype.report = function() {

};

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* var obj = new MovieRentingSystem(n, entries)
* var param_1 = obj.search(movie)
* obj.rent(shop,movie)
* obj.drop(shop,movie)
* var param_4 = obj.report()
*/
```

TypeScript Solution:

```
/**
* Problem: Design Movie Rental System
* Difficulty: Hard
* Tags: array, hash, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class MovieRentingSystem {
constructor(n: number, entries: number[][]) {

}

search(movie: number): number[] {

}

rent(shop: number, movie: number): void {

}
drop(shop: number, movie: number): void {

}

report(): number[][] {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* var obj = new MovieRentingSystem(n, entries)
* var param_1 = obj.search(movie)
* obj.rent(shop,movie)
* obj.drop(shop,movie)
* var param_4 = obj.report()
*/
```

C# Solution:

```
/*
* Problem: Design Movie Rental System
* Difficulty: Hard
* Tags: array, hash, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class MovieRentingSystem {

public MovieRentingSystem(int n, int[][] entries) {

}

public IList<int> Search(int movie) {

}
public void Rent(int shop, int movie) {

}

public void Drop(int shop, int movie) {

}

public IList<IList<int>> Report() {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* MovieRentingSystem obj = new MovieRentingSystem(n, entries);
* IList<int> param_1 = obj.Search(movie);
* obj.Rent(shop,movie);
* obj.Drop(shop,movie);
* IList<IList<int>> param_4 = obj.Report();
*/
```

C Solution:

```
/*
* Problem: Design Movie Rental System
* Difficulty: Hard
* Tags: array, hash, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} MovieRentingSystem;
```

```
MovieRentingSystem* movieRentingSystemCreate(int n, int** entries, int
entriesSize, int* entriesColSize) {

}

int* movieRentingSystemSearch(MovieRentingSystem* obj, int movie, int*
retSize) {

}

void movieRentingSystemRent(MovieRentingSystem* obj, int shop, int movie) {

}

void movieRentingSystemDrop(MovieRentingSystem* obj, int shop, int movie) {

}

int** movieRentingSystemReport(MovieRentingSystem* obj, int* retSize, int**
retColSize) {

}

void movieRentingSystemFree(MovieRentingSystem* obj) {

}

/**
* Your MovieRentingSystem struct will be instantiated and called as such:
* MovieRentingSystem* obj = movieRentingSystemCreate(n, entries, entriesSize,
entriesColSize);
* int* param_1 = movieRentingSystemSearch(obj, movie, retSize);

* movieRentingSystemRent(obj, shop, movie);

* movieRentingSystemDrop(obj, shop, movie);

* int** param_4 = movieRentingSystemReport(obj, retSize, retColSize);

* movieRentingSystemFree(obj);
*/
```

Go Solution:

```go
// Problem: Design Movie Rental System
// Difficulty: Hard
// Tags: array, hash, sort, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type MovieRentingSystem struct {

}


func Constructor(n int, entries [][]int) MovieRentingSystem {

}


func (this *MovieRentingSystem) Search(movie int) []int {

}


func (this *MovieRentingSystem) Rent(shop int, movie int) {

}


func (this *MovieRentingSystem) Drop(shop int, movie int) {

}


func (this *MovieRentingSystem) Report() [][]int {

}


/**
* Your MovieRentingSystem object will be instantiated and called as such:
```

```
* obj := Constructor(n, entries);
* param_1 := obj.Search(movie);
* obj.Rent(shop,movie);
* obj.Drop(shop,movie);
* param_4 := obj.Report();
*/
```

Kotlin Solution:

```
class MovieRentingSystem(n: Int, entries: Array<IntArray>) {

fun search(movie: Int): List<Int> {

}

fun rent(shop: Int, movie: Int) {

}

fun drop(shop: Int, movie: Int) {

}

fun report(): List<List<Int>> {

}

}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* var obj = MovieRentingSystem(n, entries)
* var param_1 = obj.search(movie)
* obj.rent(shop,movie)
* obj.drop(shop,movie)
* var param_4 = obj.report()
*/
```

Swift Solution:

```
class MovieRentingSystem {

init(_ n: Int, _ entries: [[Int]]) {

}

func search(_ movie: Int) -> [Int] {

}

func rent(_ shop: Int, _ movie: Int) {

}

func drop(_ shop: Int, _ movie: Int) {

}

func report() -> [[Int]] {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* let obj = MovieRentingSystem(n, entries)
* let ret_1: [Int] = obj.search(movie)
* obj.rent(shop, movie)
* obj.drop(shop, movie)
* let ret_4: [[Int]] = obj.report()
*/
```

Rust Solution:

```
// Problem: Design Movie Rental System
// Difficulty: Hard
// Tags: array, hash, sort, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```rust
struct MovieRentingSystem {

}



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

fn new(n: i32, entries: Vec<Vec<i32>>) -> Self {

}

fn search(&self, movie: i32) -> Vec<i32> {

}

fn rent(&self, shop: i32, movie: i32) {

}

fn drop(&self, shop: i32, movie: i32) {

}

fn report(&self) -> Vec<Vec<i32>> {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* let obj = MovieRentingSystem::new(n, entries);
* let ret_1: Vec<i32> = obj.search(movie);
* obj.rent(shop, movie);
* obj.drop(shop, movie);
* let ret_4: Vec<Vec<i32>> = obj.report();
*/
```

Ruby Solution:

```ruby
class MovieRentingSystem

=begin
:type n: Integer
:type entries: Integer[][]
=end
def initialize(n, entries)

end



=begin
:type movie: Integer
:rtype: Integer[]
=end
def search(movie)

end



=begin
:type shop: Integer
:type movie: Integer
:rtype: Void
=end
def rent(shop, movie)

end



=begin
:type shop: Integer
:type movie: Integer
:rtype: Void
=end
def drop(shop, movie)

end
```

```
=begin
:rtype: Integer[][]
=end
def report()

end



end

# Your MovieRentingSystem object will be instantiated and called as such:
# obj = MovieRentingSystem.new(n, entries)
# param_1 = obj.search(movie)
# obj.rent(shop, movie)
# obj.drop(shop, movie)
# param_4 = obj.report()
```

PHP Solution:

```
class MovieRentingSystem {
/**
* @param Integer $n
* @param Integer[][] $entries
*/
function __construct($n, $entries) {

}

/**
* @param Integer $movie
* @return Integer[]
*/
function search($movie) {

}

/**
* @param Integer $shop
* @param Integer $movie
* @return NULL
*/
function rent($shop, $movie) {

}

/**
* @param Integer $shop
* @param Integer $movie
* @return NULL
*/
function drop($shop, $movie) {

}

/**
* @return Integer[][]
*/
function report() {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* $obj = MovieRentingSystem($n, $entries);
* $ret_1 = $obj->search($movie);
* $obj->rent($shop, $movie);
* $obj->drop($shop, $movie);
* $ret_4 = $obj->report();
*/
```

Dart Solution:

```
class MovieRentingSystem {

MovieRentingSystem(int n, List<List<int>> entries) {

}

List<int> search(int movie) {

}
void rent(int shop, int movie) {

}

void drop(int shop, int movie) {

}

List<List<int>> report() {

}
}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* MovieRentingSystem obj = MovieRentingSystem(n, entries);
* List<int> param1 = obj.search(movie);
* obj.rent(shop,movie);
* obj.drop(shop,movie);
* List<List<int>> param4 = obj.report();
*/
```

Scala Solution:

```
class MovieRentingSystem(_n: Int, _entries: Array[Array[Int]]) {

def search(movie: Int): List[Int] = {

}

def rent(shop: Int, movie: Int): Unit = {

}

def drop(shop: Int, movie: Int): Unit = {

}

def report(): List[List[Int]] = {
}

}

/**
* Your MovieRentingSystem object will be instantiated and called as such:
* val obj = new MovieRentingSystem(n, entries)
* val param_1 = obj.search(movie)
* obj.rent(shop,movie)
* obj.drop(shop,movie)
* val param_4 = obj.report()
*/
```

Elixir Solution:

```
defmodule MovieRentingSystem do
@spec init_(n :: integer, entries :: [[integer]]) :: any
def init_(n, entries) do

end

@spec search(movie :: integer) :: [integer]
def search(movie) do

end

@spec rent(shop :: integer, movie :: integer) :: any
def rent(shop, movie) do

end

@spec drop(shop :: integer, movie :: integer) :: any
def drop(shop, movie) do

end

@spec report() :: [[integer]]
def report() do

end
end
```

```
# Your functions will be called as such:
# MovieRentingSystem.init_(n, entries)
# param_1 = MovieRentingSystem.search(movie)
# MovieRentingSystem.rent(shop, movie)
# MovieRentingSystem.drop(shop, movie)
# param_4 = MovieRentingSystem.report()

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

Erlang Solution:

```
-spec movie_renting_system_init_(N :: integer(), Entries :: [[integer()]]) ->
any().
movie_renting_system_init_(N, Entries) ->
.

-spec movie_renting_system_search(Movie :: integer()) -> [integer()].
movie_renting_system_search(Movie) ->
.

-spec movie_renting_system_rent(Shop :: integer(), Movie :: integer()) ->
any().
movie_renting_system_rent(Shop, Movie) ->
.

-spec movie_renting_system_drop(Shop :: integer(), Movie :: integer()) ->
any().
movie_renting_system_drop(Shop, Movie) ->
.

-spec movie_renting_system_report() -> [[integer()]].
movie_renting_system_report() ->
.


%% Your functions will be called as such:
%% movie_renting_system_init_(N, Entries),
%% Param_1 = movie_renting_system_search(Movie),
%% movie_renting_system_rent(Shop, Movie),
%% movie_renting_system_drop(Shop, Movie),
%% Param_4 = movie_renting_system_report(),

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

Racket Solution:

```
(define movie-renting-system%
(class object%
(super-new)

; n : exact-integer?
; entries : (listof (listof exact-integer?))
(init-field
n
entries)

; search : exact-integer? -> (listof exact-integer?)
(define/public (search movie)
)
; rent : exact-integer? exact-integer? -> void?
(define/public (rent shop movie)
)
; drop : exact-integer? exact-integer? -> void?
(define/public (drop shop movie)
)
; report : -> (listof (listof exact-integer?))
(define/public (report)
)))

;; Your movie-renting-system% object will be instantiated and called as such:
;; (define obj (new movie-renting-system% [n n] [entries entries]))
;; (define param_1 (send obj search movie))
;; (send obj rent shop movie)
;; (send obj drop shop movie)
;; (define param_4 (send obj report))
```