Problem 3408: Design Task Manager

Problem Information

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

Problem Description

There is a task management system that allows users to manage their tasks, each associated with a priority. The system should efficiently handle adding, modifying, executing, and removing tasks.

Implement the

TaskManager

class:

TaskManager(vector<vector<int>>& tasks)

initializes the task manager with a list of user-task-priority triples. Each element in the input list is of the form

[userId, taskId, priority]

, which adds a task to the specified user with the given priority.

void add(int userId, int taskId, int priority)

adds a task with the specified

taskId

and

priority

to the user with

userId

. It is

guaranteed

that

taskId

does not

exist

in the system.

void edit(int taskId, int newPriority)

updates the priority of the existing

taskId

to

newPriority

. It is

guaranteed

that

taskId

exists

in the system.

void rmv(int taskId)

removes the task identified by

taskId

from the system. It is

guaranteed

that

taskId

exists

in the system.

int execTop()

executes the task with the

highest

priority across all users. If there are multiple tasks with the same

highest

priority, execute the one with the highest

taskId

. After executing, the

taskId

is

removed

from the system. Return the

userId

associated with the executed task. If no tasks are available, return -1.

Note

that a user may be assigned multiple tasks.

Example 1:

Input:

["TaskManager", "add", "edit", "execTop", "rmv", "add", "execTop"]

[[[[1, 101, 10], [2, 102, 20], [3, 103, 15]]], [4, 104, 5], [102, 8], [], [101], [5, 105, 15], []]

Output:

[null, null, null, 3, null, null, 5]

Explanation

TaskManager taskManager = new TaskManager([[1, 101, 10], [2, 102, 20], [3, 103, 15]]); // Initializes with three tasks for Users 1, 2, and 3.

taskManager.add(4, 104, 5); // Adds task 104 with priority 5 for User 4.

taskManager.edit(102, 8); // Updates priority of task 102 to 8.

taskManager.execTop(); // return 3. Executes task 103 for User 3.

taskManager.rmv(101); // Removes task 101 from the system.

taskManager.add(5, 105, 15); // Adds task 105 with priority 15 for User 5.

taskManager.execTop(); // return 5. Executes task 105 for User 5.

Constraints:

1 <= tasks.length <= 10

5

0 <= userId <= 10

5

0 <= taskId <= 10

5

0 <= priority <= 10

9

0 <= newPriority <= 10

9

At most

2 * 10

5

calls will be made in

total

to

add

,

edit

,

rmv

, and

execTop

methods.

The input is generated such that

taskId

will be valid.

Code Snippets

C++:

```
class TaskManager {
public:
TaskManager(vector<vector<int>>& tasks) {

}

void add(int userId, int taskId, int priority) {

}

void edit(int taskId, int newPriority) {

}
void rmv(int taskId) {

}

int execTop() {

}
};

/**
* Your TaskManager object will be instantiated and called as such:
* TaskManager* obj = new TaskManager(tasks);
* obj->add(userId,taskId,priority);
* obj->edit(taskId,newPriority);
* obj->rmv(taskId);
* int param_4 = obj->execTop();
*/
```

Java:

```
class TaskManager {

public TaskManager(List<List<Integer>> tasks) {

}

public void add(int userId, int taskId, int priority) {

}

public void edit(int taskId, int newPriority) {

}

public void rmv(int taskId) {

}

public int execTop() {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* TaskManager obj = new TaskManager(tasks);
* obj.add(userId,taskId,priority);
* obj.edit(taskId,newPriority);
* obj.rmv(taskId);
* int param_4 = obj.execTop();
*/
```

Python3:

```
class TaskManager:

def __init__(self, tasks: List[List[int]]):



def add(self, userId: int, taskId: int, priority: int) -> None:



def edit(self, taskId: int, newPriority: int) -> None:



def rmv(self, taskId: int) -> None:



def execTop(self) -> int:




# Your TaskManager object will be instantiated and called as such:
# obj = TaskManager(tasks)
# obj.add(userId,taskId,priority)
# obj.edit(taskId,newPriority)
# obj.rmv(taskId)
# param_4 = obj.execTop()
```

Python:

```
class TaskManager(object):
```

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


def add(self, userId, taskId, priority):
"""
:type userId: int
:type taskId: int
:type priority: int
:rtype: None
"""


def edit(self, taskId, newPriority):
"""
:type taskId: int
:type newPriority: int
:rtype: None
"""


def rmv(self, taskId):
"""
:type taskId: int
:rtype: None
"""


def execTop(self):
"""
:rtype: int
"""



# Your TaskManager object will be instantiated and called as such:
# obj = TaskManager(tasks)
# obj.add(userId,taskId,priority)
# obj.edit(taskId,newPriority)
# obj.rmv(taskId)
# param_4 = obj.execTop()
```

JavaScript:

```
/**
* @param {number[][]} tasks
*/
var TaskManager = function(tasks) {

};

/**
* @param {number} userId
* @param {number} taskId
* @param {number} priority
* @return {void}
*/
TaskManager.prototype.add = function(userId, taskId, priority) {

};

/**
* @param {number} taskId
* @param {number} newPriority
* @return {void}
*/
TaskManager.prototype.edit = function(taskId, newPriority) {

};

/**
* @param {number} taskId
* @return {void}
*/
TaskManager.prototype.rmv = function(taskId) {

};

/**
* @return {number}
*/
TaskManager.prototype.execTop = function() {
};

/**
* Your TaskManager object will be instantiated and called as such:
* var obj = new TaskManager(tasks)
* obj.add(userId,taskId,priority)
* obj.edit(taskId,newPriority)
* obj.rmv(taskId)
* var param_4 = obj.execTop()
*/
```

TypeScript:

```
class TaskManager {
constructor(tasks: number[][]) {

}

add(userId: number, taskId: number, priority: number): void {

}

edit(taskId: number, newPriority: number): void {

}

rmv(taskId: number): void {

}

execTop(): number {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* var obj = new TaskManager(tasks)
* obj.add(userId,taskId,priority)
* obj.edit(taskId,newPriority)
* obj.rmv(taskId)
* var param_4 = obj.execTop()
*/
```

C#:

```
public class TaskManager {

public TaskManager(IList<IList<int>> tasks) {

}

public void Add(int userId, int taskId, int priority) {

}

public void Edit(int taskId, int newPriority) {

}

public void Rmv(int taskId) {

}

public int ExecTop() {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* TaskManager obj = new TaskManager(tasks);
* obj.Add(userId,taskId,priority);
* obj.Edit(taskId,newPriority);
* obj.Rmv(taskId);
* int param_4 = obj.ExecTop();
*/
```

C:

```
typedef struct {

} TaskManager;


TaskManager* taskManagerCreate(int** tasks, int tasksSize, int* tasksColSize)
{

}

void taskManagerAdd(TaskManager* obj, int userId, int taskId, int priority) {

}

void taskManagerEdit(TaskManager* obj, int taskId, int newPriority) {

}

void taskManagerRmv(TaskManager* obj, int taskId) {

}

int taskManagerExecTop(TaskManager* obj) {

}

void taskManagerFree(TaskManager* obj) {

}

/**
* Your TaskManager struct will be instantiated and called as such:
* TaskManager* obj = taskManagerCreate(tasks, tasksSize, tasksColSize);
* taskManagerAdd(obj, userId, taskId, priority);

* taskManagerEdit(obj, taskId, newPriority);

* taskManagerRmv(obj, taskId);

* int param_4 = taskManagerExecTop(obj);

* taskManagerFree(obj);
```

```
*/
```

Go:

```
type TaskManager struct {

}


func Constructor(tasks [][]int) TaskManager {

}


func (this *TaskManager) Add(userId int, taskId int, priority int) {

}


func (this *TaskManager) Edit(taskId int, newPriority int) {

}


func (this *TaskManager) Rmv(taskId int) {

}


func (this *TaskManager) ExecTop() int {

}


/**
* Your TaskManager object will be instantiated and called as such:
* obj := Constructor(tasks);
* obj.Add(userId,taskId,priority);
* obj.Edit(taskId,newPriority);
* obj.Rmv(taskId);
* param_4 := obj.ExecTop();
*/
```

Kotlin:

```
class TaskManager(tasks: List<List<Int>>) {

fun add(userId: Int, taskId: Int, priority: Int) {

}

fun edit(taskId: Int, newPriority: Int) {

}

fun rmv(taskId: Int) {

}

fun execTop(): Int {

}

}

/**
* Your TaskManager object will be instantiated and called as such:
* var obj = TaskManager(tasks)
* obj.add(userId,taskId,priority)
* obj.edit(taskId,newPriority)
* obj.rmv(taskId)
* var param_4 = obj.execTop()
*/
```

Swift:

```
class TaskManager {

init(_ tasks: [[Int]]) {

}

func add(_ userId: Int, _ taskId: Int, _ priority: Int) {
}

func edit(_ taskId: Int, _ newPriority: Int) {

}

func rmv(_ taskId: Int) {

}

func execTop() -> Int {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* let obj = TaskManager(tasks)
* obj.add(userId, taskId, priority)
* obj.edit(taskId, newPriority)
* obj.rmv(taskId)
* let ret_4: Int = obj.execTop()
*/
```

Rust:

```
struct TaskManager {

}



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

fn new(tasks: Vec<Vec<i32>>) -> Self {

}

fn add(&self, user_id: i32, task_id: i32, priority: i32) {
}

fn edit(&self, task_id: i32, new_priority: i32) {

}

fn rmv(&self, task_id: i32) {

}

fn exec_top(&self) -> i32 {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* let obj = TaskManager::new(tasks);
* obj.add(userId, taskId, priority);
* obj.edit(taskId, newPriority);
* obj.rmv(taskId);
* let ret_4: i32 = obj.exec_top();
*/
```

Ruby:

```
class TaskManager

=begin
:type tasks: Integer[][]
=end
def initialize(tasks)

end



=begin
:type user_id: Integer
:type task_id: Integer
:type priority: Integer
:rtype: Void
```

```
=end
def add(user_id, task_id, priority)

end



=begin
:type task_id: Integer
:type new_priority: Integer
:rtype: Void
=end
def edit(task_id, new_priority)

end



=begin
:type task_id: Integer
:rtype: Void
=end
def rmv(task_id)

end



=begin
:rtype: Integer
=end
def exec_top()

end



end

# Your TaskManager object will be instantiated and called as such:
# obj = TaskManager.new(tasks)
# obj.add(user_id, task_id, priority)
# obj.edit(task_id, new_priority)
# obj.rmv(task_id)
# param_4 = obj.exec_top()
```

PHP:

```
class TaskManager {
/**
* @param Integer[][] $tasks
*/
function __construct($tasks) {

}

/**
* @param Integer $userId
* @param Integer $taskId
* @param Integer $priority
* @return NULL
*/
function add($userId, $taskId, $priority) {

}

/**
* @param Integer $taskId
* @param Integer $newPriority
* @return NULL
*/
function edit($taskId, $newPriority) {

}

/**
* @param Integer $taskId
* @return NULL
*/
function rmv($taskId) {

}

/**
* @return Integer
*/
function execTop() {
}
}

/**
* Your TaskManager object will be instantiated and called as such:
* $obj = TaskManager($tasks);
* $obj->add($userId, $taskId, $priority);
* $obj->edit($taskId, $newPriority);
* $obj->rmv($taskId);
* $ret_4 = $obj->execTop();
*/
```

Dart:

```
class TaskManager {

TaskManager(List<List<int>> tasks) {

}

void add(int userId, int taskId, int priority) {

}

void edit(int taskId, int newPriority) {

}

void rmv(int taskId) {

}

int execTop() {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* TaskManager obj = TaskManager(tasks);
* obj.add(userId,taskId,priority);
* obj.edit(taskId,newPriority);
* obj.rmv(taskId);
* int param4 = obj.execTop();
*/
```

Scala:

```
class TaskManager(_tasks: List[List[Int]]) {

def add(userId: Int, taskId: Int, priority: Int): Unit = {

}

def edit(taskId: Int, newPriority: Int): Unit = {

}

def rmv(taskId: Int): Unit = {

}

def execTop(): Int = {

}

}

/**
* Your TaskManager object will be instantiated and called as such:
* val obj = new TaskManager(tasks)
* obj.add(userId,taskId,priority)
* obj.edit(taskId,newPriority)
* obj.rmv(taskId)
* val param_4 = obj.execTop()
*/
```

Elixir:

```
defmodule TaskManager do
@spec init_(tasks :: [[integer]]) :: any
def init_(tasks) do

end
```

```elixir


@spec add(user_id :: integer, task_id :: integer, priority :: integer) :: any
def add(user_id, task_id, priority) do

end

@spec edit(task_id :: integer, new_priority :: integer) :: any
def edit(task_id, new_priority) do

end

@spec rmv(task_id :: integer) :: any
def rmv(task_id) do

end

@spec exec_top() :: integer
def exec_top() do

end
end

# Your functions will be called as such:
# TaskManager.init_(tasks)
# TaskManager.add(user_id, task_id, priority)
# TaskManager.edit(task_id, new_priority)
# TaskManager.rmv(task_id)
# param_4 = TaskManager.exec_top()

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

Erlang:

```erlang
-spec task_manager_init_(Tasks :: [[integer()]]) -> any().
task_manager_init_(Tasks) ->
.

-spec task_manager_add(UserId :: integer(), TaskId :: integer(), Priority ::
integer()) -> any().
task_manager_add(UserId, TaskId, Priority) ->
.
```

```
-spec task_manager_edit(TaskId :: integer(), NewPriority :: integer()) ->
any().
task_manager_edit(TaskId, NewPriority) ->
.

-spec task_manager_rmv(TaskId :: integer()) -> any().
task_manager_rmv(TaskId) ->
.

-spec task_manager_exec_top() -> integer().
task_manager_exec_top() ->
.


%% Your functions will be called as such:
%% task_manager_init_(Tasks),
%% task_manager_add(UserId, TaskId, Priority),
%% task_manager_edit(TaskId, NewPriority),
%% task_manager_rmv(TaskId),
%% Param_4 = task_manager_exec_top(),

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

Racket:

```
(define task-manager%
(class object%
(super-new)

; tasks : (listof (listof exact-integer?))
(init-field
tasks)

; add : exact-integer? exact-integer? exact-integer? -> void?
(define/public (add user-id task-id priority)
)
; edit : exact-integer? exact-integer? -> void?
(define/public (edit task-id new-priority)
)
; rmv : exact-integer? -> void?
(define/public (rmv task-id)
)
; exec-top : -> exact-integer?
(define/public (exec-top)
)))

;; Your task-manager% object will be instantiated and called as such:
;; (define obj (new task-manager% [tasks tasks]))
;; (send obj add user-id task-id priority)
;; (send obj edit task-id new-priority)
;; (send obj rmv task-id)
;; (define param_4 (send obj exec-top))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Task Manager
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class TaskManager {
public:
TaskManager(vector<vector<int>>& tasks) {

}

void add(int userId, int taskId, int priority) {

}

void edit(int taskId, int newPriority) {

}


void rmv(int taskId) {

}

int execTop() {

}
};

/**
* Your TaskManager object will be instantiated and called as such:
* TaskManager* obj = new TaskManager(tasks);
* obj->add(userId,taskId,priority);
* obj->edit(taskId,newPriority);
* obj->rmv(taskId);
* int param_4 = obj->execTop();
*/
```

Java Solution:

```
/**
* Problem: Design Task Manager
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class TaskManager {

public TaskManager(List<List<Integer>> tasks) {

}

public void add(int userId, int taskId, int priority) {

}
public void edit(int taskId, int newPriority) {

}

public void rmv(int taskId) {

}

public int execTop() {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* TaskManager obj = new TaskManager(tasks);
* obj.add(userId,taskId,priority);
* obj.edit(taskId,newPriority);
* obj.rmv(taskId);
* int param_4 = obj.execTop();
*/
```

Python3 Solution:

```
"""
Problem: Design Task Manager
Difficulty: Medium
Tags: hash, queue, heap

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

class TaskManager:

def __init__(self, tasks: List[List[int]]):



def add(self, userId: int, taskId: int, priority: int) -> None:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class TaskManager(object):

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


def add(self, userId, taskId, priority):
"""
:type userId: int
:type taskId: int
:type priority: int
:rtype: None
"""


def edit(self, taskId, newPriority):
"""
:type taskId: int
:type newPriority: int
:rtype: None
"""


def rmv(self, taskId):
"""
:type taskId: int
:rtype: None
"""


def execTop(self):
"""
:rtype: int
"""
```

```
# Your TaskManager object will be instantiated and called as such:
# obj = TaskManager(tasks)
# obj.add(userId,taskId,priority)
# obj.edit(taskId,newPriority)
# obj.rmv(taskId)
# param_4 = obj.execTop()
```

JavaScript Solution:

```
/**
* Problem: Design Task Manager
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[][]} tasks
*/
var TaskManager = function(tasks) {

};

/**
* @param {number} userId
* @param {number} taskId
* @param {number} priority
* @return {void}
*/
TaskManager.prototype.add = function(userId, taskId, priority) {

};

/**
* @param {number} taskId
* @param {number} newPriority
* @return {void}
*/
TaskManager.prototype.edit = function(taskId, newPriority) {

};

/**
* @param {number} taskId
* @return {void}
*/
TaskManager.prototype.rmv = function(taskId) {

};

/**
* @return {number}
*/
TaskManager.prototype.execTop = function() {

};

/**
* Your TaskManager object will be instantiated and called as such:
* var obj = new TaskManager(tasks)
* obj.add(userId,taskId,priority)
* obj.edit(taskId,newPriority)
* obj.rmv(taskId)
* var param_4 = obj.execTop()
*/
```

TypeScript Solution:

```
/**
* Problem: Design Task Manager
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/
```

```
class TaskManager {
constructor(tasks: number[][]) {

}

add(userId: number, taskId: number, priority: number): void {

}

edit(taskId: number, newPriority: number): void {

}

rmv(taskId: number): void {

}

execTop(): number {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* var obj = new TaskManager(tasks)
* obj.add(userId,taskId,priority)
* obj.edit(taskId,newPriority)
* obj.rmv(taskId)
* var param_4 = obj.execTop()
*/
```

C# Solution:

```
/*
* Problem: Design Task Manager
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

public class TaskManager {

public TaskManager(IList<IList<int>> tasks) {

}

public void Add(int userId, int taskId, int priority) {

}

public void Edit(int taskId, int newPriority) {

}

public void Rmv(int taskId) {

}

public int ExecTop() {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* TaskManager obj = new TaskManager(tasks);
* obj.Add(userId,taskId,priority);
* obj.Edit(taskId,newPriority);
* obj.Rmv(taskId);
* int param_4 = obj.ExecTop();
*/
```

C Solution:

```
/*
* Problem: Design Task Manager
* Difficulty: Medium
* Tags: hash, queue, heap
```

```
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/




typedef struct {

} TaskManager;


TaskManager* taskManagerCreate(int** tasks, int tasksSize, int* tasksColSize)
{

}

void taskManagerAdd(TaskManager* obj, int userId, int taskId, int priority) {

}

void taskManagerEdit(TaskManager* obj, int taskId, int newPriority) {

}

void taskManagerRmv(TaskManager* obj, int taskId) {

}

int taskManagerExecTop(TaskManager* obj) {

}

void taskManagerFree(TaskManager* obj) {

}

/**
* Your TaskManager struct will be instantiated and called as such:
* TaskManager* obj = taskManagerCreate(tasks, tasksSize, tasksColSize);
* taskManagerAdd(obj, userId, taskId, priority);

* taskManagerEdit(obj, taskId, newPriority);

* taskManagerRmv(obj, taskId);

* int param_4 = taskManagerExecTop(obj);

* taskManagerFree(obj);
*/
```

Go Solution:

```go
// Problem: Design Task Manager
// Difficulty: Medium
// Tags: hash, queue, heap
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

type TaskManager struct {

}


func Constructor(tasks [][]int) TaskManager {

}


func (this *TaskManager) Add(userId int, taskId int, priority int) {

}


func (this *TaskManager) Edit(taskId int, newPriority int) {

}
```

```
func (this *TaskManager) Rmv(taskId int) {

}


func (this *TaskManager) ExecTop() int {

}


/**
* Your TaskManager object will be instantiated and called as such:
* obj := Constructor(tasks);
* obj.Add(userId,taskId,priority);
* obj.Edit(taskId,newPriority);
* obj.Rmv(taskId);
* param_4 := obj.ExecTop();
*/
```

Kotlin Solution:

```
class TaskManager(tasks: List<List<Int>>) {

fun add(userId: Int, taskId: Int, priority: Int) {

}

fun edit(taskId: Int, newPriority: Int) {

}

fun rmv(taskId: Int) {

}

fun execTop(): Int {

}

}
```

```
/**
* Your TaskManager object will be instantiated and called as such:
* var obj = TaskManager(tasks)
* obj.add(userId,taskId,priority)
* obj.edit(taskId,newPriority)
* obj.rmv(taskId)
* var param_4 = obj.execTop()
*/
```

Swift Solution:

```
class TaskManager {

init(_ tasks: [[Int]]) {

}

func add(_ userId: Int, _ taskId: Int, _ priority: Int) {

}

func edit(_ taskId: Int, _ newPriority: Int) {

}

func rmv(_ taskId: Int) {

}

func execTop() -> Int {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* let obj = TaskManager(tasks)
* obj.add(userId, taskId, priority)
* obj.edit(taskId, newPriority)
* obj.rmv(taskId)
* let ret_4: Int = obj.execTop()
*/
```

Rust Solution:

```
// Problem: Design Task Manager
// Difficulty: Medium
// Tags: hash, queue, heap
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

struct TaskManager {

}



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

fn new(tasks: Vec<Vec<i32>>) -> Self {

}

fn add(&self, user_id: i32, task_id: i32, priority: i32) {

}

fn edit(&self, task_id: i32, new_priority: i32) {

}

fn rmv(&self, task_id: i32) {

}

fn exec_top(&self) -> i32 {
}
}

/**
* Your TaskManager object will be instantiated and called as such:
* let obj = TaskManager::new(tasks);
* obj.add(userId, taskId, priority);
* obj.edit(taskId, newPriority);
* obj.rmv(taskId);
* let ret_4: i32 = obj.exec_top();
*/
```

Ruby Solution:

```
class TaskManager

=begin
:type tasks: Integer[][]
=end
def initialize(tasks)

end



=begin
:type user_id: Integer
:type task_id: Integer
:type priority: Integer
:rtype: Void
=end
def add(user_id, task_id, priority)

end



=begin
:type task_id: Integer
:type new_priority: Integer
:rtype: Void
=end
```

```
def edit(task_id, new_priority)

end



=begin
:type task_id: Integer
:rtype: Void
=end
def rmv(task_id)

end



=begin
:rtype: Integer
=end
def exec_top()

end



end

# Your TaskManager object will be instantiated and called as such:
# obj = TaskManager.new(tasks)
# obj.add(user_id, task_id, priority)
# obj.edit(task_id, new_priority)
# obj.rmv(task_id)
# param_4 = obj.exec_top()
```

PHP Solution:

```
class TaskManager {
/**
* @param Integer[][] $tasks
*/
function __construct($tasks) {

}
/**
* @param Integer $userId
* @param Integer $taskId
* @param Integer $priority
* @return NULL
*/
function add($userId, $taskId, $priority) {

}

/**
* @param Integer $taskId
* @param Integer $newPriority
* @return NULL
*/
function edit($taskId, $newPriority) {

}

/**
* @param Integer $taskId
* @return NULL
*/
function rmv($taskId) {

}

/**
* @return Integer
*/
function execTop() {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* $obj = TaskManager($tasks);
* $obj->add($userId, $taskId, $priority);
* $obj->edit($taskId, $newPriority);
* $obj->rmv($taskId);
* $ret_4 = $obj->execTop();
```

```
*/
```

Dart Solution:

```
class TaskManager {

TaskManager(List<List<int>> tasks) {

}

void add(int userId, int taskId, int priority) {

}

void edit(int taskId, int newPriority) {

}

void rmv(int taskId) {

}

int execTop() {

}
}

/**
* Your TaskManager object will be instantiated and called as such:
* TaskManager obj = TaskManager(tasks);
* obj.add(userId,taskId,priority);
* obj.edit(taskId,newPriority);
* obj.rmv(taskId);
* int param4 = obj.execTop();
*/
```

Scala Solution:

```
class TaskManager(_tasks: List[List[Int]]) {

def add(userId: Int, taskId: Int, priority: Int): Unit = {
}

def edit(taskId: Int, newPriority: Int): Unit = {

}

def rmv(taskId: Int): Unit = {

}

def execTop(): Int = {

}

}

/**
* Your TaskManager object will be instantiated and called as such:
* val obj = new TaskManager(tasks)
* obj.add(userId,taskId,priority)
* obj.edit(taskId,newPriority)
* obj.rmv(taskId)
* val param_4 = obj.execTop()
*/
```

Elixir Solution:

```
defmodule TaskManager do
@spec init_(tasks :: [[integer]]) :: any
def init_(tasks) do

end

@spec add(user_id :: integer, task_id :: integer, priority :: integer) :: any
def add(user_id, task_id, priority) do

end

@spec edit(task_id :: integer, new_priority :: integer) :: any
def edit(task_id, new_priority) do
```

```


end

@spec rmv(task_id :: integer) :: any
def rmv(task_id) do

end

@spec exec_top() :: integer
def exec_top() do

end
end

# Your functions will be called as such:
# TaskManager.init_(tasks)
# TaskManager.add(user_id, task_id, priority)
# TaskManager.edit(task_id, new_priority)
# TaskManager.rmv(task_id)
# param_4 = TaskManager.exec_top()

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

Erlang Solution:

```
-spec task_manager_init_(Tasks :: [[integer()]]) -> any().
task_manager_init_(Tasks) ->
.

-spec task_manager_add(UserId :: integer(), TaskId :: integer(), Priority ::
integer()) -> any().
task_manager_add(UserId, TaskId, Priority) ->
.

-spec task_manager_edit(TaskId :: integer(), NewPriority :: integer()) ->
any().
task_manager_edit(TaskId, NewPriority) ->
.

-spec task_manager_rmv(TaskId :: integer()) -> any().
```

```
task_manager_rmv(TaskId) ->
.

-spec task_manager_exec_top() -> integer().
task_manager_exec_top() ->
.



%% Your functions will be called as such:
%% task_manager_init_(Tasks),
%% task_manager_add(UserId, TaskId, Priority),
%% task_manager_edit(TaskId, NewPriority),
%% task_manager_rmv(TaskId),
%% Param_4 = task_manager_exec_top(),

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

Racket Solution:

```
(define task-manager%
(class object%
(super-new)

; tasks : (listof (listof exact-integer?))
(init-field
tasks)

; add : exact-integer? exact-integer? exact-integer? -> void?
(define/public (add user-id task-id priority)
)
; edit : exact-integer? exact-integer? -> void?
(define/public (edit task-id new-priority)
)
; rmv : exact-integer? -> void?
(define/public (rmv task-id)
)
; exec-top : -> exact-integer?
(define/public (exec-top)
)))
```

```
;; Your task-manager% object will be instantiated and called as such:
;; (define obj (new task-manager% [tasks tasks]))
;; (send obj add user-id task-id priority)
;; (send obj edit task-id new-priority)
;; (send obj rmv task-id)
;; (define param_4 (send obj exec-top))
```