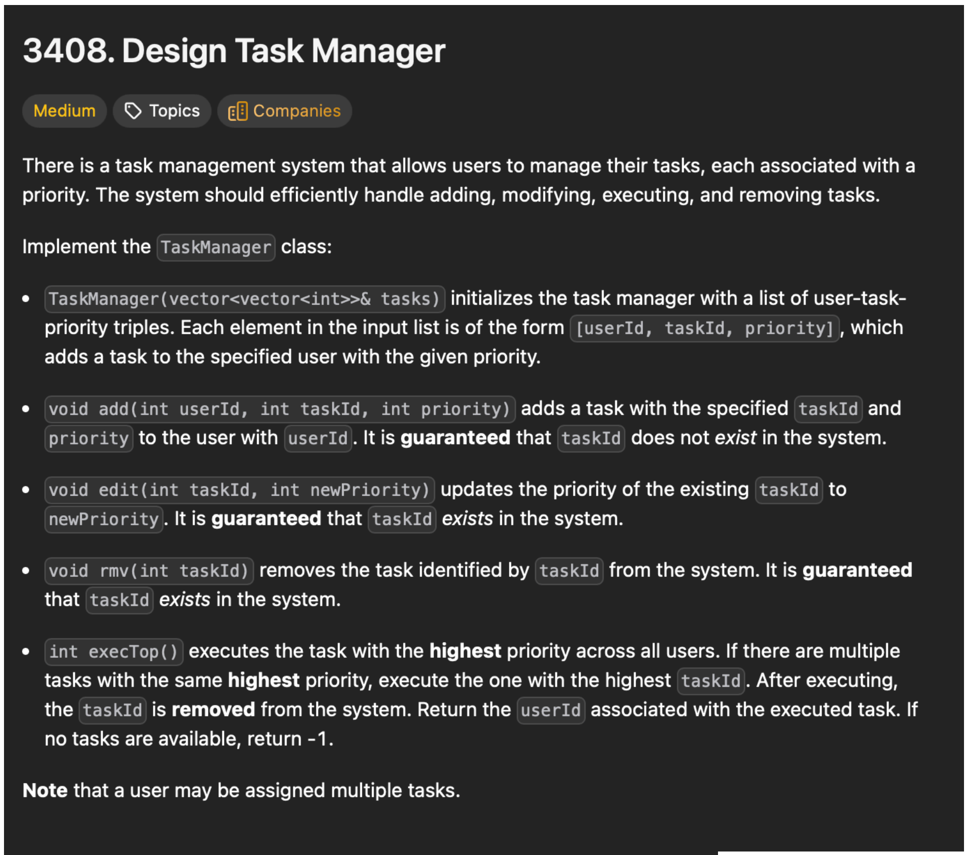


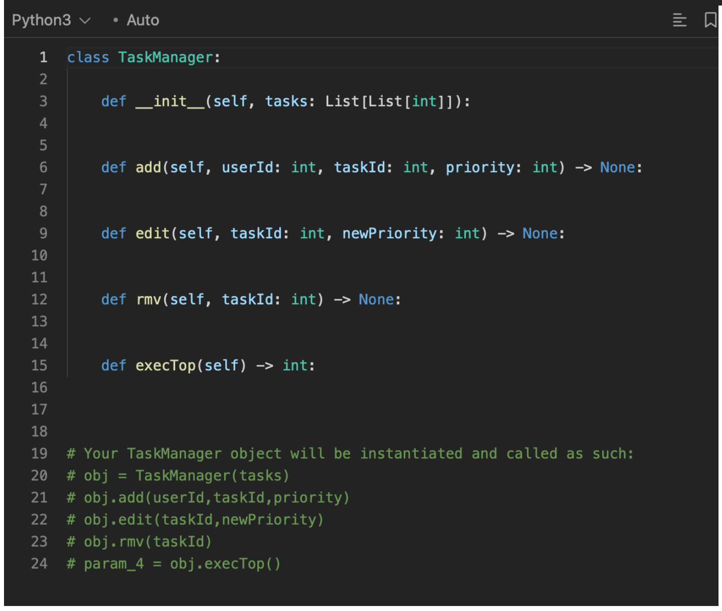
3408. Design Task Manager

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.

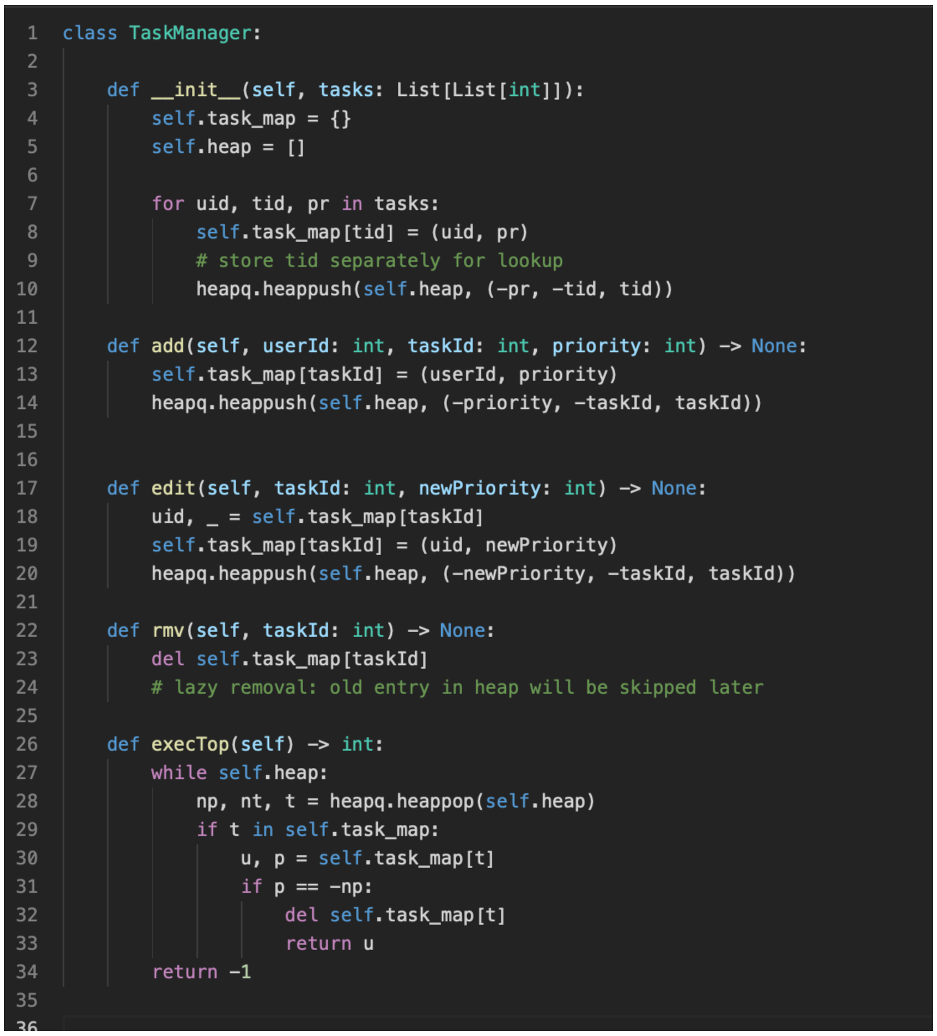
Note that a user may be assigned multiple tasks.



```python
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()
```

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

,[2,102,9],[3,103,5] [4,104,5], [50,101,8]



```python
class TaskManager:

    def __init__(self, tasks: List[List[int]]):
        self.task_map = {}
        self.heap = []

        for uid, tid, pr in tasks:
            self.task_map[tid] = (uid, pr)
            # store tid separately for lookup
            heapq.heappush(self.heap, (-pr, -tid, tid))

    def add(self, userId: int, taskId: int, priority: int) -> None:
        self.task_map[taskId] = (userId, priority)
        heapq.heappush(self.heap, (-priority, -taskId, taskId))


    def edit(self, taskId: int, newPriority: int) -> None:
        uid, _ = self.task_map[taskId]
        self.task_map[taskId] = (uid, newPriority)
        heapq.heappush(self.heap, (-newPriority, -taskId, taskId))

    def rmv(self, taskId: int) -> None:
        del self.task_map[taskId]
        # lazy removal: old entry in heap will be skipped later

    def execTop(self) -> int:
        while self.heap:
            np, nt, t = heapq.heappop(self.heap)
            if t in self.task_map:
                u, p = self.task_map[t]
                if p == -np:
                    del self.task_map[t]
                    return u
        return -1
```

task_map: taskId -> (userId, priority)
heap: (-p, -taskId, taskId)
to make sure first heap sort by priority then taskId, the higher the priority and taskId, comes earlier (priority val larger, taskId val smaller)

first get, then overwrite

del

the taskId being called will be removed after call, so a del is involved in the loop