**Complexity Analysis**

* Daniel José Plazas (A00400085)
* Sebastián Erazo (A00400086)
* Juan Manuel Casanova (A00400090)

**First Method:**

public void undoAction(){

NodeStack<Task> action = actions.pop(0);

if(action!=null){

switch (action.getAction()){

case "added":

tasks.remove(action.getS().getTitle());

if(action.getS().getPriority()==0){

nonPriorityTasks.remove(action.getS());

}else{

priorityTasks.remove(action.getS());

}

break;

case "removed":

if(action.getT()!=null){

tasks.put(action.getT().getTitle(),action.getT());

if(action.getT().getPriority()==0){

nonPriorityTasks.add(action.getT());

}else{

priorityTasks.add(action.getT());

}

}

break;

case "modified":

if(action.getT()!=null && action.getS()!=null){

System.out.println("T==S? = " + (action.getS()==action.getT()));

tasks.remove(action.getS().getTitle());

tasks.put(action.getT().getTitle(),action.getT());

if(action.getT().getPriority()==0){

nonPriorityTasks.remove(action.getS());

nonPriorityTasks.add(action.getT());

}else{

priorityTasks.remove(action.getS());

priorityTasks.add(action.getT());

}

}

break;

}

}

}

**Time Complexity**

The time complexity of the method is O(1), because any case of the switch takes 1 time to do it. so, in general, the complexity would be O(1).

**Spatial Complexity**

In spatial complexity is the same as time complexity if enters in a case where the nonPriorityTask is added, or O(1) if the task is added in a priorityTask. Why? This is because the method nonPriorityTask.add() takes just O(1), but priorityTask.add() takes O(n). So:

1. O(1) if the system add a nonPriorityTask
2. O(n) if the system adds a priorityTask

-------------------------------------------------------------------------------------------------------------------------

**Second Method**

public String modifyTask(String title, String newTitle, String newDescription,Calendar deadline, int newPriority,int option) {

String message = "The task was modified successfully!";

Task oldTask = null;

Task newTask = null;

try {

if (tasks.isEmpty()) {

throw new HashTableIsEmptyException("The task list is empty.");

}

if (!tasks.containsKey(title)) {

throw new NonExistentKeyException("Task with title '" + title + "' does not exist.");

}

oldTask = (Task) tasks.getValue(title).clone();

newTask = (Task)tasks.getValue(title).clone();

tasks.remove(title);

switch (option) {

case 1:

newTask.setTitle(newTitle);

break;

case 2:

newTask.setDescription(newDescription);

break;

case 3:

newTask.setDeadline(deadline);

break;

case 4:

newTask.setPriority(newPriority);

break;

}

tasks.put(newTask.getTitle(), newTask);

// Update priority/non-priority task lists

if (oldTask.getPriority() == 0) {

nonPriorityTasks.remove(oldTask);

nonPriorityTasks.add(newTask);

} else {

priorityTasks.remove(oldTask);

priorityTasks.add(newTask);

}

//System.out.println("Old==New? : " + (oldTask==newTask));

} catch (HashTableIsEmptyException e) {

message = e.getMessage();

} catch (NonExistentKeyException e1) {

message = e1.getMessage();

}

actions.push(oldTask,"modified",newTask);

return message;

}

**Time Complexity**

Overall, the time complexity of the modifyTask method is O(1) because it contains a series of constant-time operations, and the number of operations doesn't depend on the size of the data structures.

**Spatial Complexity**

This method is the same case of the previous method (undoAction). If the system adds a nonPriorityTask, the complexity would be O(1) and if the system adds a priorityTask, it would be O(n).