**SPACE AND TEMPORARY COMPLEXITY – IPS Software**

* **Temporary Complexity**

**Method Undo**

public String undo(){

Action action = stack.pop().getValue(); **// 1**

if(action!=null){ **// 1**

if(action.getWhichAction().equalsIgnoreCase("entry")){ **// 1**

if(action.getWhichPatient().getPriority()>0){ **// 1**

if(action.getLab()==1){ **// 1**

for (int i = 0; i < priorityQueueHematology.getPriorityQueue().length; i++) { **// n**

if (priorityQueueHematology.getPriorityQueue()[i].getElement().getId().equalsIgnoreCase

(action.getWhichPatient().getId())) { **// n-1**

priorityQueueHematology.getPriorityQueue()[i].getElement().setInQueue(false); // **n-1**

priorityQueueHematology.delete(i); **// 2(n - 1) - 1 = 2n - 2 - 1 = | 2n - 3 |**

return action.getWhichAction(); **// n-1**

}

} // delete an inserted patient priority queue

} else {

for (int i = 0; i < priorityQueueGeneralPurpose.getPriorityQueue().length; i++) {

if (priorityQueueGeneralPurpose.getPriorityQueue()[i].getElement().getId().equalsIgnoreCase

(action.getWhichPatient().getId())) {

priorityQueueGeneralPurpose.getPriorityQueue()[i].getElement().setInQueue(false);

priorityQueueGeneralPurpose.delete(i);

return action.getWhichAction();

}

} // delete an inserted patient priority queue

}

} else {

if(action.getLab()==1){

for (int i = 0; i < queueHematology.getQueue().length; i++) {

if (queueHematology.getQueue()[i].getValue().getId().equalsIgnoreCase

(action.getWhichPatient().getId())) {

queueHematology.getQueue()[i].getValue().setInQueue(false);

queueHematology.delete(i);

return action.getWhichAction();

}

} // delete an inserted patient queue

} else {

for (int i = 0; i < queueGeneralPurpose.getQueue().length; i++) { // n

if (queueGeneralPurpose.getQueue()[i].getValue().getId().equalsIgnoreCase

(action.getWhichPatient().getId())) { // n - 1

queueGeneralPurpose.getQueue()[i].getValue().setInQueue(false); // n - 1

queueGeneralPurpose.delete(i); // 2n - 3

return action.getWhichAction(); // n-1

}

} // delete an inserted patient queue

}

}

} else {

if(action.getWhichPatient().getPriority()>0){

if(action.getLab()==1){

priorityQueueHematology.insert(new PriorityQueueNode<>(action.getWhichPatient(), action.getWhichPatient().getPriority()));

// insert a deleted patient priority queue

action.getWhichPatient().setInQueue(true);

return action.getWhichAction(); // 1

} else {

priorityQueueGeneralPurpose.insert(new PriorityQueueNode<>(action.getWhichPatient(), action.getWhichPatient().getPriority()));

// insert a deleted patient priority queue

action.getWhichPatient().setInQueue(true);

return action.getWhichAction(); // 1

}

} else {

if(action.getLab()==1){

queueHematology.offer(new QueueNode<>(action.getWhichPatient()));

// insert a deleted patient queue

action.getWhichPatient().setInQueue(true);

return action.getWhichAction(); // 1

} else {

queueGeneralPurpose.offer(new QueueNode<>(action.getWhichPatient()));

// insert a deleted patient queue

action.getWhichPatient().setInQueue(true);

return action.getWhichAction(); // 1

}

}

}

}

return "";

} // undo

Ya que en gran cantidad del método hay estructuras de if-else, tomamos solo la parte de ella más grande (con mayor complejidad)

**Complejidad algorítmica TOTAL**

**1+1+1+1+1+n+n-1+n-1+2n-3+n-1 = 6n - 1**

**Méthod out**

public String out(){

Patient patient; **// 1**

for (int i = 0; i < priorityQueueHematology.getPriorityQueue().length; i++) { **// n**

if(priorityQueueHematology.getPriorityQueue()[i]!=null){ **// n-1**

if(priorityQueueHematology.getPriorityQueue()[i].getKey()==0){ **// n-1**

QueueNode<Patient> queueNode = new QueueNode<>(priorityQueueHematology.getPriorityQueue()[i].getElement()); **// n-1**

queueHematology.offer(queueNode); **// n-1**

priorityQueueHematology.delete(i); **// 3n-3**

}

}

}

for (int i = 0; i < queueHematology.size(); i++) { **// n**

if(queueHematology.getQueue()[i]!=null){ **// n-1**

if(queueHematology.getQueue()[i].getValue().getPriority()>0){ **//n-1**

PriorityQueueNode<Patient, Integer> priorityQueueNode = new PriorityQueueNode<>(queueHematology.getQueue()[i].getValue(), queueHematology.getQueue()[i].getValue().getPriority()); **//n-1**

priorityQueueHematology.insert(priorityQueueNode); **// n-1**

queueHematology.delete(i); **//3n-3**

}

}

}

if(priorityQueueHematology.isEmpty()){ **// 1**

patient = queueHematology.poll(); **// 1**

patient.setInQueue(false); **// 1**

} else {

patient = priorityQueueHematology.extractMax();

patient.setInQueue(false);

}

stack.push(new Action("out", patient, 1)); **// 1**

if(patient!=null){ **// 1**

return "- - ID: "+patient.getId()+ **// 1**

"\n- - NAME: "+patient.getName();

} else {

return null;

}

} // out

Ya que en algunas partes del método hay estructuras de if-else, tomamos solo la parte de ella más grande (con mayor complejidad)

**Complejidad algorítmica TOTAL**

**1+n+n-1+n-1+3n-3+3n-3+n+n-1+n-1+n-1+1+1+1+1+1+1 = 13n – 3**

* **Space Complexity**

**Method Undo**

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **VARIABLE** | **LENGHT OF 1 ATOMIC VALUE** | **AMOUNT OF ATOMIC VALUES** |
| **INPUT** | **None** | **0 bits** | **0** |
| **AUXILARY** | **i** | **32** | **1** |
| **OUTPUT** | **undo** | **32** | **1** |

**Auxiliar space complexity: None**

**Output + auxiliar space complexity = 1 + 1 =** θ(2)

**Total space complexity = 0 + 1 + 1 = θ(2)**

**Method out**

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **VARIABLE** | **LENGHT OF 1 ATOMIC VALUE** | **AMOUNT OF ATOMIC VALUES** |
| **INPUT** | **None** | **0 bits** | **0** |
| **AUXILARY** | **i** | **32 bits** | **1** |
| **OUTPUT** | **patient** | **32 bits** | **1** |

**Auxiliar space complexity: None**

**Output + auxiliar space complexity = 1 + 1 =** θ(2)

**Total space complexity = 0 + 1 + 1 = θ(2)**