Data Structure Design

CAA2: Second continuous assessment test

Statement

In this CAA we are going to continue with the design of the data structure of the fictitious application intended for Technology and Telecommunications Center (CTT). We will expand the functionalities to be able to work with nonlinear structures, which will allow us to use larger volumes of information.

Following the launch of the application, certain issues and deficiencies have been brought to our attention through user complaints, encompassing both companies and workers. Despite initial assumptions that specific features would cater to a limited user base, the application has generated more interest than anticipated. In response, the CTT has entrusted us with a set of modifications to address these issues and enhance the overall functionality of the application.

To begin with, we will integrate the capability to manage CTT rooms, including the materials available and assigned to each room. This enhancement aims to provide better control and prevent complaints arising from material shortages. Additionally, to ensure the optimal condition of the rooms, the CTT has requested that the application be equipped to manage its personnel undertaking various tasks (security, cleaning, maintenance, etc.) in each room.

A recurrent concern voiced by workers pertains to the registration process. Securing work opportunities becomes challenging if one is not vigilant enough with job offers. While the CTT believes in rewarding efficiency, it is not contemplating an overhaul of the registration system. However, it deems it prudent to refine the substitute registration process to prioritize individuals with less work experience. To achieve this, a tiered level system will be implemented, categorizing workers as follows:

- **Expert** : Workers who have accumulated 1000 or more hours.
- **Senior** : Workers with 500 or more accumulated hours and less than 1000.
- **Junior** : Workers with 200 or more accumulated hours and less than 500.
- **Intern** : Workers with 10 or more accumulated hours and less than 200.
- **Beginner** : Workers with less than 10 accumulated hours.

NOTE: Each accumulated day of work is equivalent to 8 hours

In addition to everything indicated above, to carry out this activity consider the following aspects:

Modifications regarding CAA1:

- The number of **workers** will be very large and will increase.
- The number of **job offers** will be very large and is expected to increase.
- The number of **companies** will be large but stable.
- At the request of the CTT, the operation “consult the best valued job offer” will not return a single offer, but the 10 best valued offers.

CAA2 news:

- The number of **rooms (R)** will be very large and known.
- The number of **employees (E)** of the CTT will be known and large.
- The number of **roles (R)** will be small, determined to be a few dozen.
- The number of **employees who have a role** (**ER**) is very large and indeterminate.
- The number of **materials (M)** will be very large and will grow over time.

NOTE:

- The starting point for developing this CAA is the **official solution of CAA1** published in the classroom.
- In general, in all activities it is necessary to optimize the efficiency of query operations even if this penalizes write operations.
- If there is any error condition that is not indicated in the statement and you think it is necessary to add it, you can add it.

Exercise 1 (2.5 points)

Specify the **CTTCompaniesJobs ADT** that allows the following operations:

CAA1 operations that must modify their behavior:

NOTE: From CAA1 you should review that, although the signature of the operations does not change, the implementation may do so due to the new choice of data structures:

1. **Create job request** : The behavior is as described in CAA1, but the requested room is provided in this case.
2. **Issue a result to a request** : The behavior is as described in CAA1 but taking into account that a room cannot be used at the same time for two offers. If an attempt is made to approve a request in which the room is already being used, the error will be reported.
3. **Sign up for an offer by a worker** : The behavior of this operation will be similar to that specified in CAA1 but taking into account the criteria established when managing substitutes.
4. **Consult the 10 best-rated offers** : If there is no offer, an error should be indicated.

Operations added in CAA2:

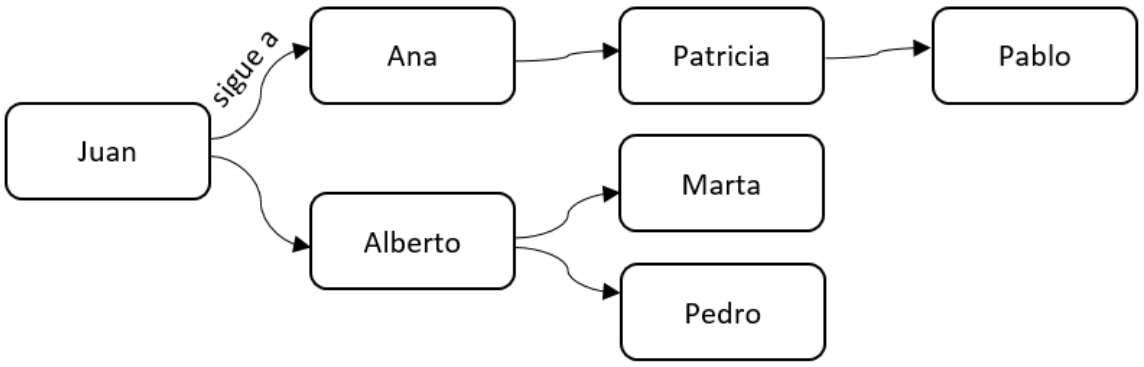
1. **Add a role** : For each role we will know its identifier and name. If a role with this identifier already exists, we update its data.
2. **Add an employee** : For each employee we will know their ID, their first name, their last name, date of birth and their role. It is known that the role indicated is among those existing in the system. If there is already an employee with that ID, we update their data.
3. **Add a room** : For each room, its identifier, name, description and type (laboratory, office or coworking) are known. If a room with that identifier already exists, its data is updated.
4. **Assign an employee to a room** : If the room or employee does not exist, an error should be reported. An error will also be reported if the employee is already assigned to that room. An employee may be assigned to more than one room at a time.
5. **Query employees assigned to a room** : Returns all employees in a room. If the room does not exist or there are no employees, an error must be reported.
6. **Query all employees with a certain role:** It is known that the role exists in the system. If there are no employees with this role, an error must be reported.
7. **Add material** : For each material we know its identifier, name and description. If a material with that identifier already exists, its data is updated.
8. **Assign material to a room** : If the room or material does not exist, an error should be reported. An error will also be reported if the material is already assigned to that room. If the material was already assigned to a different room, it will be detached from it and assigned to the new one.
9. **Query the level of a worker** : Returns the current level of the worker. If the worker does not exist, the error must be indicated.

10. **Consult the workers registered for an offer (not substitutes)** : Returns all workers registered for the offer. If the offer does not exist or there are no registered workers, an error will be indicated.
11. **Consult the substitute workers registered for an offer** : Returns all the substitute workers for an offer. If the offer does not exist or there are no substitute workers, an error will be indicated.
12. **Consult rooms without assigned employees** : If there are no rooms and/or all have assigned employees, the error must be indicated.
13. **Consult the 5 best prepared rooms (with the most material)** : If there is no room and/or none have material, the error must be indicated.

As a measure to facilitate communication and foster camaraderie among employees, the CTT has requested to include some social functions in the application. Specifically, employees will be able to follow each other in a similar way as on any social network.

For the following operations, we will consider that employees can follow each other (be followers and followed) on the social network.

14. **Add an employee as a follower** of **another** . If the employee to be added or the employee who receives a new follower does not exist, an error will be indicated. It is assumed that there is no prior relationship between employees.
15. **Consult the list of followers** (followers). Provides an employee's employee followers. If the employee does not exist, the error must be indicated. If there are no follower employees, the error must be indicated.
16. **View the list of employees you follow.** Provides the list of employees that a given employee follows. If the employee does not exist, the error must be indicated. If there are no employees you follow, the error must be indicated.
17. **Suggest employees to follow** : Given an employee, the list of employees who are followed by those followed by the given employee is provided. If the employee does not exist or there are no followers, the error must be indicated. For example, in the figure below, the suggestion about Juan would return Patricia, Martha, and Pedro.



18. **Show employees assigned to the same rooms and who are not followed:** Provides a list of employees assigned to the same rooms as a given employee and who are not among the employee's followers. If the employee does not exist, the error must be indicated. If there are no employees still assigned to the same rooms, the error must be indicated.

Section a) (1 point)

Specify the **signature** of the **CTTCompaniesJobs ADT**. That is, indicate the name of the operations described above. Also indicate what the input parameters are and the type of data returned in each case.

1. addRole(roleId, description)
2. addEmployee(dni, name, surname, birthday, roleId)
3. addRoom(roomId, name, description, type)
4. assignEmployee(dni, roomId)
5. getEmployeesByRoom(roomId): Iterador
6. getEmployeesByRole(roleId): Iterador
7. addEquipment(equipmentId, name, description)
8. assignEquipment(equipmentId, roomId)
9. getLevel(workerId): Level
10. getWorkersByJobOffer(jobOfferId): Iterator
11. getSubstitutesByJobOffer(jobOfferId): Iterator
12. getRoomsWithoutEmployees(): Iterator
13. best5EquippedRooms(): Iterador
14. addFollower(employeeDni, employeeFollowerDni)
15. getFollowers(employeeDni): Iterador
16. getFollowings(employeeDni): Iterador
17. recommendations(employeeDni): Iterador
18. getUnfollowedColleagues(employeeDni): Iterador

Section b) (1.5 points)

Carry out the **contractual specification** of the operations of the ADT **CTTCompaniesJobs**. Specifically, as in CAA1, the initial conditions that the operations must have (@pre) and the conditions on the state (@post) that remains in the system (variables, data structures, return values) must be specified. after its execution. Take as a reference the specification of section 1.2.3 of Module 1 of the teaching materials. It will be valued that the specification is:

- Concise (absence of redundant or unnecessary elements).
- Precise (correct definition of the result of operations).
- Complete (consideration of all possible cases in which each operation can be executed).
- And does not contain ambiguities (exact knowledge of how each operation behaves in all possible cases).

NOTE: It is important to use a conditional description and not a procedural one, although it is not always easy to distinguish between both descriptions, which is why this aspect is mentioned.

For example, the following is the conditional (correct) description of starting a vehicle:

@pre The vehicle's engine is off.
@post The vehicle's engine is running.

However, a procedural description (incorrect for this section) could be:

@pre The vehicle's engine should be off if it shouldn't be.
@post The engine cranks until it starts to run.

To describe the ADT, it must also be taken into account that a contract should have an invariant if necessary.
Solution

@pre true.
@post if the role code is new, the roles will remain the same plus a new role with the indicated data. If not, the role data will have been updated with the new data.

1. **addRole(roleId, description)**

@pre the role exists.
@post if the employee's ID is new, the employees will remain the same plus a new employee; the number of employees will be the same plus one, and the number of employees in a role will be the same plus one. If not, the employee data will have been updated with the new data, and if the role is modified, the number of employees in the old role will be the same minus one, and the number of employees in the new role will be the same plus one.

2. **addEmployee(dni, name, surname, birthday, roleId)**

@pre true.
@post if the roomId is new, the rooms will remain the same plus a new room with the indicated data. If not, the room data will have been updated with the new data.

3. **addRoom(roomId, name, description, type)**

@pre true.
@post if the employee is not assigned to the room, the employees in the room will remain the same plus a new one. If the employee is already assigned to the room, an error will be displayed. If the employee or the room does not exist, an error will be indicated.

4. **assignEmployee(dni, roomId)**

@pre true.
@post returns an iterator to traverse all employees assigned to a room. In case there are no employees or the room does not exist, an error should be indicated.

5. **getEmployeesByRoom(roomId): Iterator**

@pre the role exists.
@post returns an iterator to traverse all employees of a role. If there are no employees with that role, an error should be indicated.

6. **getEmployeesByRole(roleId): Iterator**

@pre true.
@post if the equipmentId is new, the equipments will remain the same plus a new equipment with the indicated data. If not, the equipment data will have been updated with the new data.

7. **addEquipment(equipmentId, name, description)**

@pre true.
@post if the equipment is not assigned to the room, the equipments in the room will remain the same plus a new one. If the equipment is already assigned to the room, an error will be displayed. If the equipment is already assigned to another room, it will be unlinked from it and assigned to the new one, and the number of equipments in the old room is the same minus one, and the number of equipments in the new assigned room is the same plus one. If the equipment or the room does not exist, an error will be indicated.

8. **assignEquipment(equipmentId, roomId)**

@pre true.
@post returns the level associated with the worker. If the worker does not exist, an error should be indicated.

9. **getLevel(workerId): Level**

@pre true.
@post returns an iterator to traverse all non-substitute workers enrolled in the job offer jobOfferId. If the job offer does not exist or no worker is enrolled, an error should be indicated.

10. **getWorkersByJobOffer(jobOfferId): Iterator**

@pre true.
@post returns an iterator to traverse all substitute workers enrolled in the job offer jobOfferId. If the job offer does not exist or no substitute is enrolled, an error should be indicated.

11. **getSubstitutesByJobOffer(jobOfferId): Iterator**

@pre true.
@post returns an iterator to traverse all rooms that have no assigned employees. If there are no rooms without any employees, an error should be indicated.

12. **getRoomsWithoutEmployees(): Iterator**

@pre true.
@post returns an iterator to traverse the top 5 rooms with the most assigned materials. If there are no rooms or no materials assigned, an error will be indicated.

13. **best5EquippedRooms**(): Iterator

@pre employeeDni is not a follower of employeeFollowerDni.
@post the number of followers of employeeDni will be the same plus one, and the number of employees followed (followings) by employeeFollowerDni will be the same plus one. In case the employee to follow or the followed employee does not exist, an error will be indicated.

14. addFollower(employeeDni, employeeFollowerDni)

@pre true.
@post returns an iterator to traverse the followers of an employee. If the employee does not exist or has no followers, an error should be indicated.

15. getFollowers(employeeDni): Iterator

@pre true.
@post returns an iterator to traverse the employees that an employee is following. If the employee does not exist or has no employees following, an error should be indicated.

16. getFollowings(employeeDni): Iterator

@pre true.
@post returns an iterator to traverse suggested employees who are followers of employees that employeeDni is following and who are currently not followers. If the employee does not exist or has no employees following, an error will be indicated.

17. recommendations(employeeDni): Iterator

@pre true.
@post returns an iterator to traverse employees working in the same rooms as employeeDni and whom he/she is not following. If the employee does not exist or there are no employees not followed assigned to the same rooms, an error will be indicated.

18. getUnfollowedColleagues(employeeDni): Iterator

Exercise 2 (3.5 points)

Next, we will **design the associated data structures** , based on the **CTTCompaniesJobs ADT specification** , taking into account the volumes of information and the restrictive specifications described in the statement. The system should be as efficient as possible both temporally and spatially.

To make this design, take into account only the operations that are requested in the statement.

Section a) (0.25 points)

We hesitate between using a vector, an AVL, a linked list or a hash table to store the **workers .** Justify which you think is the best option.

The best option is an AVL since the number of workers becomes very large and unlimited. With a list, search operations would be linear, and with large volumes of information, it is not acceptable. A hash table is also not suitable if the information we want to store is not limited. We choose an AVL, which will provide logarithmic costs for queries and allow us to work with very large and unlimited volumes of information

Section b) (0.25 points)

What structure would be ideal for storing **substitute entries** : a queue, a stack or a list? Justify which you think is the best option.

As substitute registrations must be stored according to a specific criterion, the criterion for substitute registrations is based on the worker's level, leading us to choose a priority queue

Section c) (0.25 points)

What structure would be ideal to store CTT **employees , an AVL, a linked list or a hash table?** Justify which you think is the best option and what specific structure you would use taking into account that the searches will be carried out by ID .

As the volume of data is large, we discard using a vector or list, and we must choose between an AVL tree and a hash table. Since we are informed that the number of employees is known and large, we will choose a hash table to achieve constant query access through the employee's ID.

Section d) (0.25 points)

What data structure would you use to store the **rooms** ? And for the **equipments** ? Justify which you think is the best option in each case.

Solution

- Rooms: We know that, like employees, the number of organizing entities will be very large and known. With this information, we can determine that the best choice would be a hash table, which would guarantee efficient access.

- Equipments: A hash table would allow us to achieve constant query access even when working with very large data volumes, as would be the case with materials. However, we know that the number of these equipments is indeterminate and therefore would not be a good choice. The best option for storing equipments is an AVL tree, which will provide logarithmic costs for queries and allow us to work with very large and unlimited volumes of information

Section e) (1 point)

Justify **each and every one of the data structures to store the rest of the elements** , taking into account everything discussed in the statement and in the description of the operations. Only for mandatory operations (1-13). To do this, it is necessary to use the following format:

"To save XXX we choose an ordered linked list since the number of elements is not very large, and we need ordered traversals."

Solution

For companies, we will choose a hash table since the container size will be large but stable, and efficient access is desired.

For requests, we choose an unbounded queue since they are taken in the order of arrival, and their size is indeterminate.

For job offers, we will use an AVL tree since it will be very large and expected to grow.

For job offers where a worker is hired, we select a linked list as the number is small and indeterminate.

To store roles, we will use a Java array (Object[]).

To store employees associated with a role, we will use a linked list.

To store employees assigned to a room, we will select a linked list.

To store materials assigned to a room, we will select a linked list.

To store the rooms to which a material is assigned, we will use a pointer.

To store rooms without assigned employees, we will use a linked list since their number is not known in advance.

For evaluations of a job offer, we use a linked list since it is a small and indeterminate number.

For worker registrations for a job offer, we use an unbounded queue since they are processed in the order of arrival.

For the total number of requests, we use an integer.

For the total number of rejected requests, we use an integer.

For the most active worker, we choose a pointer.

To store the top 10 rated job offers, we will use an ordered vector.

To store the top 5 best-equipped rooms, we will use an ordered vector.

Section f) (0.5 point)

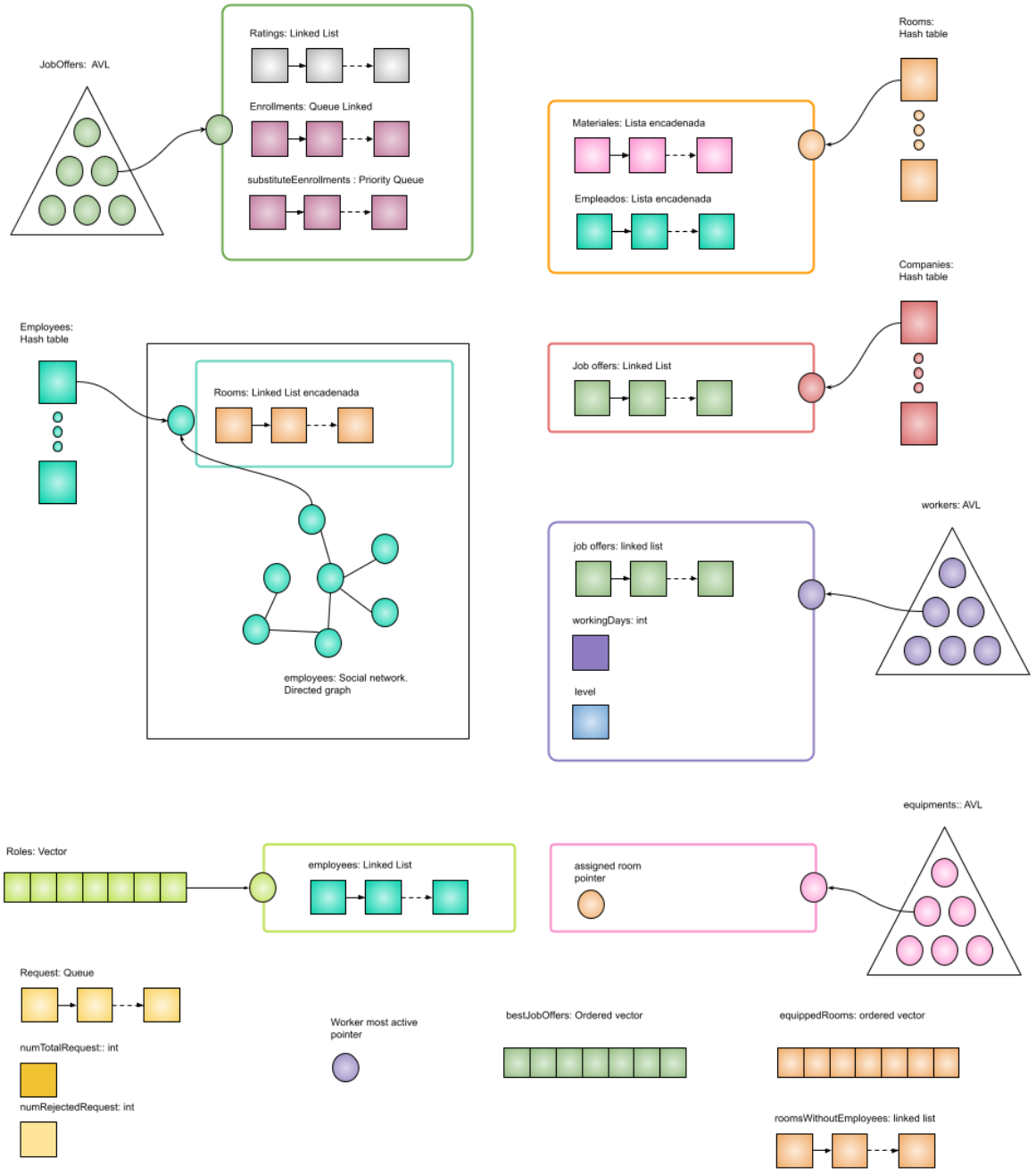
Indicate additional data structures to implement **operations** (14-18) justifying your choice briefly (max 2 paragraphs) .

Solution

To store the relationship between followers and followed employees, a directed graph will be used. To provide the list of employees assigned to the same rooms and those who are not followed, it is necessary for the employee to maintain the list of rooms to which they are assigned through a linked list. The alternative would be to traverse all rooms and the employees assigned to these rooms, which would not be scalable

Section g) (1 point)

Make a **graphical representation** of the global data structure by the **CTTCompaniesJobs ADT** where the data structures chosen to represent each of the parts and the relationships between them are seen. You must represent the complete structure, with all the structures that allow you to implement the operations defined in the specification.



Exercise 3 (3 points)

In exercise 1 we have defined the specification of the **CTTCompaniesJobs ADT** with its operations, and in exercise 2 we have chosen the data structures to implement each part of the ADT. In this exercise we ask you to look at the algorithms that will be used to implement some of the operations specified above and the study of their **efficiency .** Keep in mind that the implementation of the operations is closely linked to the choice of data structures you have made.

Section a) (1.5 points)

Describe and carry out the efficiency study for the following operations:

- Add a room.
- Assign material to a room.
- Assign an employee to a room.

To do so, you must briefly describe the behavior of the operation, indicating the steps that comprise it. You can use pseudocode or phrases like: "insert into vector / delete from linked list / query element in sorted linked list /...", indicating the asymptotic efficiency of each step and calculating the total efficiency of the operation.

Solution

Add a room:

Search for the room in the CTT's room hash table => O(1).

Insert the room into the CTT's room hash table => O(1).

Total: **O(1)**.

Assign equipment to a room:

Search for the equipment in the AVL of equipment => O(log E).

Search for the room in the CTT's room hash table => O(1).

Add the equipment to the linked list of equipment in a room => O(1).

Update the pointer of the room assigned to the equipment => O(1).

Total: **O(log E)**.

Assign an employee to a room:

Search for the room in the CTT's room hash table => O(1).

Search for the employee in the CTT's employee hash table => O(1).

Search for the employee in the linked list of employees in a room => O(ER).

Delete the employee in the linked list of employees in a room => O(1).

Add the employee to the linked list of employees in a room => O(1).

Total: **O(ER)**.

Note: In case the room to which the employee has been assigned was previously empty, it would be necessary to search in the list of empty rooms O(ER) and delete it O(1). In case the original room becomes empty when abandoned by the employee, it would be necessary to insert it into the list of empty rooms O(1)

Section b) (1.5 points)

Currently, no operation allows a worker to withdraw from an offer, so registrations and substitute registrations never decrease. In a real application it would be essential to have this functionality. Describe in detail what the behavior of said deregistration operation would be like.

Solution

Firstly, it would be necessary to check whether the job offer exists. To do this, we must traverse the AVL of job offers. If the offer is not found, we should report an error.

Next, we should verify that the worker who is resigning from a specific job offer indeed has a registration in that offer. To do this, we must traverse the unbounded queue of registrations and, if not found there, also the queue of substitute registrations. If the worker's registration is not found in the offer, we should report an error. Otherwise, two situations could arise:

- If the registration is in the unbounded queue of registrations: In this case, upon finding the registration in this queue, we would need to:

- Search for the job offer in the linked list of the worker.
- Remove the job offer from the linked list of the worker.
- Remove the worker's registration from the unbounded queue of the job offer.
- Extract the first registration from the priority queue of substitute registrations.
- Insert the registration into the unbounded queue of the job offer.
- Add the offer to the linked list of job offers for the worker.

If the registration is in the priority queue of substitute registrations: In this case, it would be sufficient to remove the registration from the priority queue.

Exercise 4 (1 point)

Section a) [0.5 point]

Indicate which **ADT from the ADT library** https://eimtgit.uoc.edu/DS/DSLib seem more suitable for use in the implementation of each of the data structures defined by the **CTTCompaniesJobs ADT** for operations 1-13 **.** If there is no implementation already made in the ADT library of the subject, briefly indicate how you would implement it.

Solution

Workers: **DictionaryAVLImpl**.

Job Offers of a Company: **Linked List**.

Companies: **Hash Table**.

Requests: **QueueLinkedList**.

Job Offers: **DictionaryAVLImpl**.

Job Offers of a Worker: **LinkedList**.

Ratings: **LinkedLis**t.

Registrations: **QueueLinkedList**.

Substitute Registrations: **PriorityQueue**.

Employees: **HashTable**.

Rooms: **HashTable**.

Equipments: **DictionaryAVLImp**l.

Employees Assigned to a Room: **LinkedList**.

Equipments Assigned to a Room: **LinkedList**.

Room where a Equipment is Assigned: **Pointer** - Room.

Rooms without Employees: **LinkedLis**t.

Roles: **Java Array** - Roles[].

Employees Associated with a Role: **LinkedLis**t.

Total Requests: **Integer**.

Total Rejected Requests: **Intege**r.

Most Active Worker: **Pointe**r - Worker.

Top-Rated Job Offer: **OrderedVector**.

Best-Equipped Rooms: **OrderedVector.**

Section b) 0.5

Indicate which ADT from the subject's ADT library you think is most appropriate for use in the implementation of each of the data structures defined by the CTTCompaniesJobs ADT **for** operations options 14-18 **.** If there is no implementation already made in the ADT library of the subject, briefly comment on how you would implement it.

Solution

Rooms Assigned to an Employee: **Linked List**
Social Network among Employees: **DirectedGraphImpl.**