The National Higher School of Artificial Intelligence

Data Structures and Algorithms

Project Report: Hospital Care Unit

Team Members:

- BOUBEKEUR Farida (team leader).
- BOUDAOUD Amira.
- MERS Wafaa.

Table of Content:

Keywords:

- ICU: Intensive Care Unit
- CSU: Critical State Unit

1. Explanation of the problem Treated:

Each service in a hospital has an intensive care unit, where a small number of patients is being treated (for example: 6 patients in the hospital of Beni Messous, and 3 patients in the hospital of Tizi Ouzou, CHU Nedir Mouhamed). The patients there get a special treatment due to their severe states. Each patient gets a constant record of the condition of his blood pressure and his heart rate (and a number of other measurements) in addition to a constant monitoring done by doctors. Since the intensive care unit can't support a large number of patients, the time a patient can spend there doesn't usually exceed 24h. (Boudaoud Manel & Mers Chaima, medical students, personal communication, October 29, 2022).

Our System controls and monitors what is happening in each intensive care unit in a given hospital. It gives the access to the director to monitor all the units. And it gives the access to the doctors and nurses, in each service, to add or release patients, and to check the information of any patient under treatment. This system can also cyclically print the information about the condition of the patient (This feature should be used in a separate screen for each one of the patients).

Patients can be released from an intensive care unit for one of the three reasons: their condition gets better so the doctor decides to allow them to leave the hospital, they pass away, or they are moved to a critical state unit where they receive a less intensive care but for an extended period of time.

2. The Data Chosen for Testing:

We chose a hospital that has 3 services: cardiology service, pneumology service, and gynaecology service. (We could add other services, but we kept these for the simplification of the testing phase).

Cardiology service:

We chose to run the following instructions:

- Add patient 1: Amira Boudaoud, 18 years old and track his status.
- Add patient 2: Wafaa Mers 19 years old and track its status.
- Release patient 2 for the reason 'pass away'.
- Add patient 3: Farida Boubekeur 19 years old and track his status.
- Add patient 4: Abdelhak Chellal 24 years old and track his status.
- Add patient 5: Mohcen Chouireb 22 years old and track his status.
- Release patient 5 for the reason 'allowed to leave'.
- Add patient 6: Abderrahim Rezki 19 years old and track his status.
- Add patient 7: Meriem Djidjeli 21 years old and track his status.
- Access the information of patient 3.
- Add patient 8: Sarra Arab 30 years old and track his status.
- Add patient 9: Rayane Yadel 20 years old and track his status.
- Release patient 8 for the reason 'pass away'.
- Release patient 9 for the reason 'pass away'.
- Add patient 10: Yasmina Medjri 30 years old and track his status.
- Add patient 11: Yasmine Kaced 39 years old and track his status.
- Release patient 7 for the reason 'moved to CSU'.
- Release patient 4 for the reason 'allowed to go'.
- Add patient 12: Larbi Saidchikh 70 years old and track his status.

- Release patient 6 for the reason ‘pass away’.
- Release patient 12 for the reason ‘pass away’.
- Release patient 3 for the reason ‘moved to CSU’.
- Add patient 13: Ania Boubekeur 14 years old and track his status.
- Release patient 5 for the reason ‘allowed to go’.
- Release patient 1 for the reason ‘pass away’.

Pneumology service:

We chose to run the following instructions:

- Add patient 1: Manel Boudaoud 22 and track her status.
- Add patient 2: Chaima Mers 23 and track her status.
- Release patient 2 for the reason ‘moved to CSU’.

Gynaecology service:

We chose to run the following instructions:

- Add patient 1: Farida Tounsi 50.
- Add patient 2: Dalila Sami 51.
- Add patient 3: Wissam Boudaoud 19.
- Release patient 2 for the reason ‘allowed to go’.

We followed the instructions stated above to test our program and test the worst case of having a total of 13 patients in the same service (cardiology in this case). We also used this data set to get the results of calculating the average time of the operation admit a patient.

The website of health.ucdavis.edu contains that the records of blood pressure are presented as systolic/diastolic. Normal values are the ones around 120/80 mmHg (millimeters of Mercury). Pre-hypertensive values are from 120/80 to 139/89, and hypertensive values are higher than 140/90.

The values of heart rate of patient in intensive care units can vary from 3.21 to 66.7 during death episode, and from 15.2 to 74.2 during No-death episode. (Mohammad Karimi Moridani & Seyed Kamaledin Setarehdan & Ali Motie Nasrabadi & Esmaeil Hajinasrollahm, 2014, p5).

We used the previously stated data to implement a random generation for blood pressure and heart rate. We didn’t use a previously prepared data for patients because our program runs at the same time when the data is recorded.

3. Explanation of the Design Choice:

3.1. Classes Used in the Implementation:

Our implementation is based on 4 classes: class patient, class intensive care unit, class critical state unit, and class ICU hash table that is a re-implementation of the hash table (class BSTree for part B of the project, or class AvlTree for part C).

3.1.1. Class Patient: (Header file name: Patient.h, and implementation file name: Patient.cpp)

The objects of this class represent the patients themselves. Each patient has a first name, a last name, an age, a unique ID, and a file that is created once the patient is added to the system. The file holds the name “Patient” followed by the unique ID of the patient. And it serves as an archive of the

personal information of the patient, and the data about his blood pressure and his heart rate that is being recorded while he gets the treatment in the corresponding intensive care unit.

In our case, we can't access the measurement devices of the actual intensive care unit to get the values of blood pressure and heart rate of the patients. Thus, we chose to use a random generator that takes a random value of blood pressure in the interval [90/60 , 220/120], and takes a random value of heart rate in the interval [3 , 74]. Obviously, we are aware that this randomly generated data isn't a reliable source and cannot be considered as a data base for an actual experiment, but it has been chosen just for the sake of testing and can be replaced in case the system will be implemented in real life.

In addition to the necessary setters and getters, and the functions overloading the necessary operators, this class has the following methods:

- "determineID", that is a utility function used to generate the unique ID.
- "showPatientData", that prints all what has been recorded in the file corresponding to the patient.
- "addPatientFile", that prints and enters the personal information of the patient into his file.

3.1.2. Class Intensive Care Unit: (Header file name: Intensive_care_unit.h, and implementation file name: Intensive_care_unit.cpp)

The objects of this class represent the intensive care units of each service. In part A, this class is basically a hash table of patients, considering the unique IDs as the keys. In part B, this class is a binary search tree. And in part C, this class is an AVL tree.

Each object of this class has a file where the history that states when a patient enters the ICU or when he is released is saved. The file's name is "ICU_history" followed by a number (1 for the service created first, 2 for the one created next, and so on…).

This class has an additional data member 'numOfDeaths' that represents the number of deaths that occurred in all the services (a static variable).

The following are the additional methods of class "IntensiveCareUnit":

- "insert", that simply admits a new patient to the intensive care unit.
- "remove", that deletes a patient from the intensive care unit and saves the reason of his release in the corresponding file. (In case of moving the patient to the critical state unit, this method inserts the patient to the object of the class critical state unit).
- "insertHistory_remove", a utility function that writes in the file of history when a patient is removed of the ICU.
- "insertHistory_insertion", a utility function that writes in the file of history when a patient is entered to the ICU.
- "patientInformation" , that prints the information of one patient.
- "ICU_history", the method that prints the content of the file of history.
- "ICU_listPatient", prints the patients that are in the intensive care unit.

3.1.3. Class Critical State Unit:

(Header file name: Critical_State_Unit.h, and implementation file name: Critical_State_Unit.cpp)

This class shares the same concept with the class intensive care unit. Thus, it is implemented using the same ADT used in the other class. However, it only has the method of inserting a patient.

3.2. Implementation of the unique ID generation:

The patient is described by a unique identification number. Once his data (first name, last name, age) is entered into the system, a file including all his information is created.

Doctors, nurses, and the director also have an ID for the purpose of logging into the system and having access to the provided features.

The IDs are generated as follows:

- The ID is 10 characters long. It is represented by an unsigned long in the implementation.

- All IDs in this hospital share 21 on the left, so they are of the form 21********.
- All patient's IDs share 13 right after 21, so they are of the form 2113******.
- All doctors' (or nurses) IDs share 12 right after 21, so they are of the form 2112******
    - In cardiology service, IDs have an additional 35 next to 2112. So they are of the form 211235****
    - In pneumology service, IDs have an additional 69 next to 2112. So they are of the form 211269****
    - In gynaecology service, IDs have an additional 88 next to 2112. So they are of the form 211288****
- The director's ID is different from all other IDs, it is 2175634298
- IDs of the patients are generated by the system. Each time a new patient is admitted, he gets an ID by incrementing the ID of the patient that got admitted last by 1.

Note: this algorithm of assigning IDs doesn't respect the security measures. It is generated this way just for the sake of simplification.

3.3. Implementation of the driver program:

The users of our program are either: doctors, nurses, or the director of the hospital. We chose to start with a login since the features available for the doctor or the nurse are different from those available for the director. The user needs to enter his ID to log into the system.

In case of the doctor or nurse, they log into one of the available services (the one they belong to). Five options appear and he/she needs to choose one of them. The first option is to add a new patient to the intensive care unit of this service. Here the program asks the doctor to enter the information about the new patient. And creates an object of him from class Patient and adds him to the hash table of the intensive care unit. The second option is to release a patient. Here, the doctor gets the list of the patients currently under treatment and he choose one of them to release by typing his ID then specifying the reason of the release. The third option is accessing the information of a certain patient. And that is the option used most in real life since the doctors need to check the states of their patients very often. Here the doctor gets the list of all patients and he selects one by typing his ID to let the program print all the information of the selected patient. The forth option is to track the status of a patient, this option is not supposed to be one of the options that are available in the screen of the doctor. Though, it should be available on its own in a separate screen for each one of the patients to keep track of his condition by showing the values of heart rate and blood pressure until he patient is released. Since this approach can't be applied in our mini-project, we chose to implement a simplification that prints those measurements in each second and stores the values in the file of the archive of the patient. This simplification does this job for a period of 20s only just for the sake of showing how does it actually work. The fifth, and last, option is to quit the system.

In the case of the director, five options as well appear to him. The first is to access the history of an intensive care unit. He first chooses a service then the history of the corresponding intensive care unit appears. The second option is getting the number of deaths that occurred in all the intensive care units. The third option is getting the list of patient in a chosen service. And the fourth option gives him

access to check the information and condition of one patient. He first choose the service to get the list of patients then he chooses the patient by specifying his ID.

3.4. Part A of the project:

3.4.1. The Graphical Representation of the most suitable ADT

The following figure shows the process of adding a new class object of class patient, and shows what happens in our program:

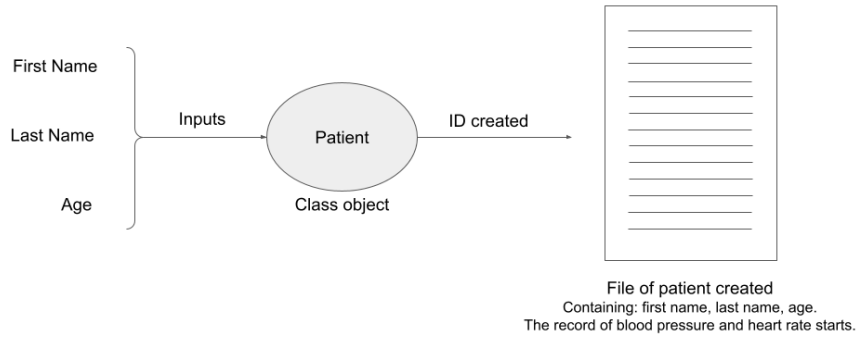

Figure 1: Graphical Representation of class patient

And the following figure shows what happens after figure 1, when a patient class object is created, it is added to the corresponding intensive care unit that is a hash table.

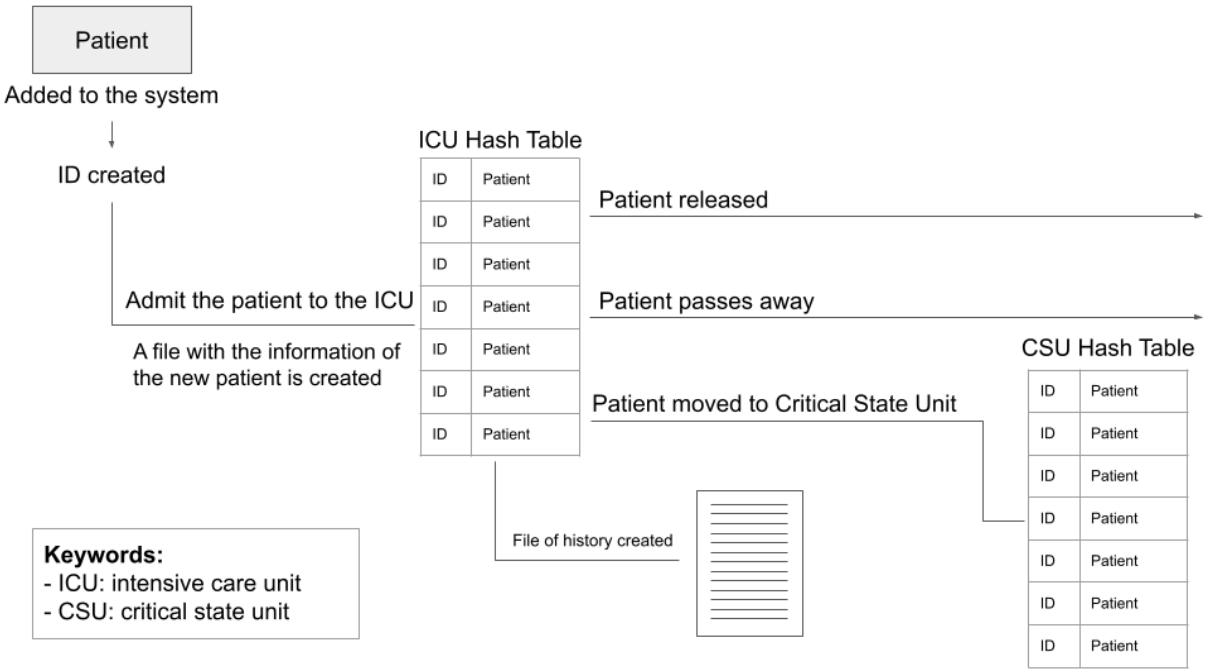

Figure 2: Graphical Representation of class intensive care unit

Figure 3 shows the process that gets repeated in the driver program and how different IDs can show different features for different users.

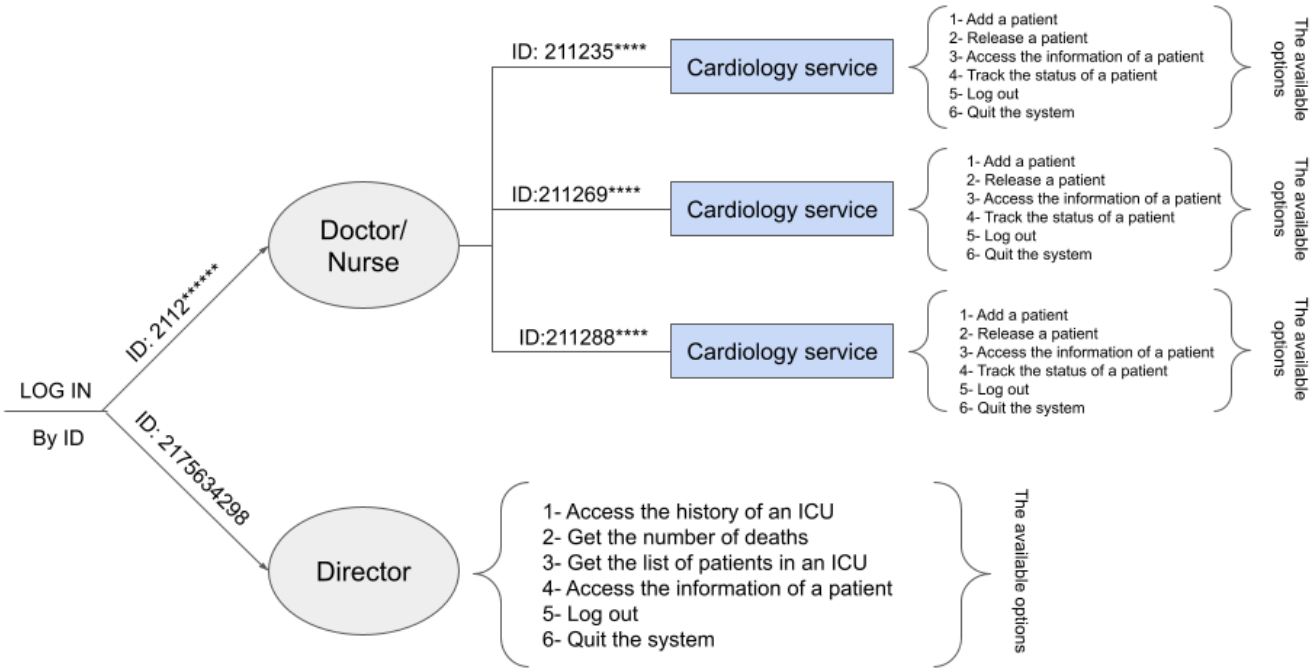

Figure 3: Graphical Representation of the driver program

Figure 4 shows how does the feature of print cyclically display the data during the execution of the program when the doctor chooses the 4th option.

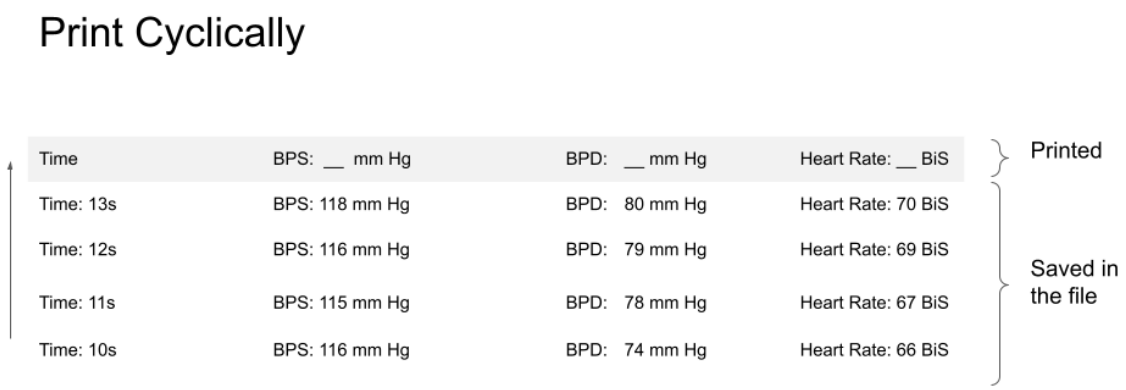

Figure 4: Graphical Representation of the feature print cyclically

3.4.1. Complete Specification of the ADT:

We used a hash table as the most suitable ADT for the monitoring system. And we implemented class 'intensive care unit' using this ADT. So the class is basically a hash table of patients, considering the unique IDs as the keys. Since our key can have reasonably many different values, choosing the most common strategy for our hash function was enough. That is *key mod tableSize*, and we ensured that the *tableSize* is always prime. To handle the collisions, we chose linear probing that is enough for our approach since the size of our table is always small. So if the blocks happen to be formed, traversing them won't take much time.

The choice of this ADT comes from the fact that accessing a specific patient using his ID is the operation used the most in our system. The hash table has a complexity of ***O(1)*** when it comes to accessing one of its elements, thus it is the best choice for our program.

Note: We re-implemented the class of hash table in the files "ICU_Hash_table.h" and "ICU_Hash_table.cpp", in order to facilitate the manipulation of this class and to add what we see fit with the other parts of our code.

3.5. Part B of the project:

In this part, we kept the same implementation as in part A, except that we replaced the hash table by a Binary Search Tree.

We added a data member to class Patient: 'key' that is generated randomly between 1 and 100, to make the comparison of patients possible. We couldn't use the unique ID for this sake because according to our implementation of the ID: Each time a patient is admitted to an intensive care unit, a new ID is generated, and it is always greater than all the previous IDs. Thus, if we use the ID to compare patients, we will always result in having a tree of N nodes with height N (degenerated tree). And each node has a left child only. Which makes the tree a linked list instead (the worst case of a BST).

3.6. Part C of the project:

The implementation of this part is too similar to the previous one. We just replaced the implementation of the BST with that of the AVL tree. The main difference between the two implementations is the balancing operation (rotations) which assures that the tree is kept balanced after each operation of admitting a patient to the intensive care unit.

Note: we re-implemented both the files of AVL tree and BST for the same reason of doing so with the hash table. And for each class, we implemented the definition and implementation of the class in the same file. We tried to separate them as we did in part A but the ADEs we used (VS code, Microsoft Visual Studio, and Code Blocks) weren't able to compile the program that way.

4. <u>The Results of the Average Time of the Operation 'Admit Patient':</u>

Here is a table that represents the average time in milliseconds of the operation that admits a new patient in the intensive care unit. These results have been taken after executing the program using the instructions mentioned in the section of 'the data chosen for testing'. Thus, the average time calculated in each case was the average of executing the function 'insert' in the intensive care unit for 18 times.

Running Time (ms)	**Part A**	**Part B**	**Part C**
Operation 1	1160	2161	1469
Operation 2	1368	1159	1639
Operation 3	1346	930	2779
Operation 4	1708	1568	9090
Operation 5	2522	811	2589
Operation 6	2052	1314	2756
Operation 7	1113	670	2177
Operation 8	1421	535	2328
Operation 9	3560	1164	2729
Operation 10	1256	691	1838
Operation 11	2308	1978	2678
Operation 12	2146	804	1604
Operation 13	34	1053	24
Operation 14	51	1778	34
Operation 15	17	1247	18
Operation 16	21	9223	21
Operation 17	10	845	10
Operation 18	122	1553	1973
Average time	**64**	**120**	**81**

The results displayed in the table show that the best running time of the operation 'admitting a new patient' was that of the hash table, then comes the results of running the operation on an AVL tree, and finally running it on the BST took the longest period of time, thus the worst result in the case of our system.

In part A, the operation of inserting an element into the hash table is of complexity *O(1).* Hence, having the best running time in this case was an inevitable result. Next, in part B, the operation of inserting an element in the Binary Search Tree is of complexity *O(N)* in the worst case (*N* is the number of nodes of the tree). That's why it took the longest running time. Last, in part C, the AVL tree is always balanced so its height is around *log N,* so accessing an element is of complexity *O(log N)* and that's why the running time of inserting into the AVL tree was better than inserting into a binary search but worse than inserting into a hash table.

5. Guidelines for using the solution:

- Once the program is executed, the login phase starts. To enter the correct ID, use the description of the implementation of the ID generation to type an ID of one of the forms mentioned there.
- You choose the format of the ID to log in according to which features you want to have access to, whether the ones of the doctor/nurse (and in which service), or those of the director.
- It is recommended to log in as a doctor/nurse first and perform a number of operation (of adding or maybe releasing), then log out and log in as a director to test the options provided for the director. Because the director monitors what is happening. So if you start by logging in as a director, you will find everything empty, and all values equal to 0 since no data was entered to the program just yet.
- When using the program as a doctor or nurse, it is recommended to use the option 'track a patient' right after adding a patient to get the file of the patient filled with the information about his condition for at least 20s.

6. Division of tasks:

We, as the team members, worked on the solution design all together. We did that during several meetings where the three of us where present to share our ideas and combine them. We did around four meetings, each one lasts around three hours to get our ideas into a shape and combine them with the results we got from doing small researches.

For the implementation, we tried to divide the tasks to do them individually, but since all the parts of the code are related to each other, we worked as a team to combine them and make several modifications to get the final version of our program. We also worked together during the phase of debugging that took us too long.

When it came to writing this part of the report we were really confused, but we tried to be as specific as possible to mention what did each one of us do even when we were working together.

We believe that this is how should a team work, because working on the project is just a phase of our process of learning, and we see that the essence is to help each other and work together to expand our knowledge.

Below is what we can call a division of tasks:

- **Farida Boubekeur:**
  - Re-implementation of class hash table from scratch
  - Re-implementation of class BST from scratch
  - Re-implementation of class AVL from scratch
  - Implementation of class Intensive Care Unit
  - Manipulation of ICU_historys' files
  - Debugging of
    - BST class
    - AVL class
    - Files' implementation
- **Amira Boudaoud:**
  - Small research about the medical information
  - Implementation of class patient
  - Implementation of class critical state unit
  - Manipulation of patients' files
  - Debugging of
    - driver program with the required changes in the classes
    - BST class
    - Files' implementation
    - Print cyclically feature
  - Testing the results

- **Wafaa Mers:**
  - Implementation of ID generation
  - Implementation of driver program (int main)
  - Debugging
    - the driver program with the required changes in the classes
    - file's implementation
    - Print cyclically feature
  - Testing the results
  - Small research about the medical information
  - Writing the report

7. References:

Heart rate: What's normal? (2022, October 8). Mayo Clinic.

https://www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979

Moridani, M. K., Setarehdan, S. K., Nasrabadi, A. M., & Hajinasrollahm, E. (2014). Analysis of heart rate variability as a predictor of mortality in cardiovascular patients of intensive care unit. *Https://Www.Sciencedirect.Com/Science/Article/Abs/Pii/S0208521615000376*.

https://sci-hub.se/https://doi.org/10.1016/j.bbe.2015.05.004

PasseportSanté : Actualité Santé, Alimentation, Exercice, Gestion Du Stress. (n.d.).

https://www.passeportsante.net/. https://www.passeportsante.net

Sports Medicine. (n.d.). *Blood Pressure | UC Davis Sports Medicine | UCDavis Health*. Copyright UC Regents. For More Information Regarding the University of California's Copyright Terms, Visit http://www.ucop.edu/services/terms.html.

https://health.ucdavis.edu/sports-medicine/resources/blood-pressure

Tension artérielle. (n.d.). Doctissimo.

https://www.doctissimo.fr/sante/dictionnaire-medical/tension-arterielle

7. Appendix:

Implementation of part A

Patient.h , Patient.cpp , Critical_State_Unit.cpp, Critical_State_Unit.h, ICU_Hash_table.h, ICU_Hash_table.cpp ,Intensive_care_unit.h. Intensive_care_unit.cpp, project_main.cpp

Implementation of part B

Binary_Search_Tree.h, Critical_State_Unit.h, intensive_care_unit.h, Patient.h, project_main.cpp, TreeNode.h

Implementation of part C

AVL_Tree.h, Critical_State_Unit.h, intensive_care_unit.h, Patient.h, project_main.cpp, TreeNode.h