Report Data Structure Project: Hospital Management System

Mr. Yotsapoom Liupolvanish	ID: 67070503493
Mr. Chaiyaphoom Chenchirotphiphat	ID: 67070503410
Mr. Atip Infa-udom	ID: 67070503446

This report is submitted as part of the coursework for CPE 112: Programming with Data Structures

Master of Engineering in Computer Engineering

Faculty of Engineering

King Mongkut's University of Technology Thonburi

Academic Year 2024

Problem Statement

Hospital can be busy, especially during an emergency.Managing the patient queue can become a real challenge for people that work in the healthcare field.Sometime people with serious conditions have to wait just like those who have minor issues.This could be bad, especially when someone life is at risk.It is important to have a better way to organize patients base on how serious their symptoms or situation are.

Our goal with the Hospital Management System is to fix this by introducing a priority tests. When a patient use this system, The system will ask for their information such as name, age, sex, phone number, any allergies they have and any underlying conditions first and ask them a few questions later like how much pain they are feeling right now, whether they feel tired or not, if they are having trouble breathing and if they are fully conscious.Based on their answers, we give them a score that helps us decide how urgent they need to be treated.Not only that but this system also provide easy use for doctors to check patients symptom at a glance by looking from the priority tests that patients have fill out.

Lastly, this system is all about making hospital queues better.It help medical staff focus on what matter most, treating patients who need urgent or critical care first while keeping everything organized and well.

Solution with functionalities

Login System

1. Main Menu: Allow user to go to the doctor or patient menu, and exit from the program.
2. Doctor Menu: Allow doctor to login as a doctor or go back to the main menu.
3. Login as Doctor: Doctor need to verify themself first by entering the username and password of the doctor correctly.
4. Patient Menu: Allow user to login as a patient, create a patient account or go back to the main menu.
5. Create a Patient Account: Allow patient to create their account by typing their name, password and password confirmation.
6. Login as Patient: Allow patient to login into their account by typing their name and password after creating their own account.
7. Back: In the Patient menu and Doctor menu is used to go back to the main menu.
8. Exit: used to exit from the program.

Doctor

1. Doctor Panel: Allows doctors to display all queues of patients, cure patients that have the highest priority, display cured patients, and go back to the doctor menu.
2. Display all queue: show all the patients queue along with their information and priority score.
3. Cure Patient: Cure patient that has the highest priority.
4. Display Cured Patients: Allows doctors to see all cured patients, to search for patients and go back.
5. All Cured Patients: Show all patients that have been cured by the doctor.
6. Search: Search name of the patient that has been cured and the doctor wants to see their information specifically.

7. Back: used to go back to the doctor panel.

Patient

1. Patient Queue System: Allows users to add queue, check queue, cancel queue and exit to go back to the patient menu.
2. Add Queue: Allows patients to write their information (such as name, age, sex, phone number, allergies and underlying conditions) that the system wants and do their priority test after writing their information.
3. Priority Test: Let patients answer each question to use for calculate the priority score, so that the doctor can know who should be cured first.
4. Check Queue: Allows patients to see their queue by typing their name in there.
5. Cancel Queue: Allows patients to cancel their queue by typing their name in there.
6. Exit: Used to go back to the patient menu.

Explanation of the chosen data structures

1. **Array**

   Why we use array

   - We use an array to add new users or patients, check if a username and password match during login or not and to show user or patient information when needed.

   Why we choose array to do this task

   - We choose arrays because it is easy to use and fast so that we can find, add or check data. Lastly, it makes our code simpler and easier to understand.

2. **Singly Linked List**

   Why we use Singly Linked List

   - We use Singly Linked List for managing the patient queue (patients that is waiting for treatment)

   Why we choose Singly Linked List to do this task

   - We choose Singly Linked List because it is efficient for insertion and deletion at the front and rear and it is ideal for queue operations where patients are add and remove frequently

3. **Priority Queue**

   Why we use Priority Queue

   - We use Priority Queue to reordering patients based on symptom severity score

   Why we choose Priority Queue to do this task:

   - We choose Priority Queue because we can ensures that critical patients or patients with more serious condition are treated first (Automatically arrange patients in order of urgency)

4. **Binary Search Tree**

   Why we use Binary Search Tree

   - We use Binary Search Tree to store record of patients who have been treated

   Why we choose Binary Search Tree to do this task:

   - We choose Binary Search Tree because it provide fast searching and insertion of patient record and can maintain sorted order for easier to get an information

Code walkthrough

1. **vari.h**

   Purpose: Store all data structures that are used in the project.

   Key Elements:

   ➔ MAX: constant defining maximum queue size.
   ➔ Patient: struct with patient information.
   ➔ Queue: struct of a patient queue with front pointer and rear index.
   ➔ node: struct for binary tree implementation.

2. **piorqueue.h**

   Purpose: Develops priority queue for patients.

   Key Functions:

   ➔ createQ(): Creates a new patient node.
   ➔ DeleteByName(): Removes a patient by name from the queue.
   ➔ Enqueue(): Adds a patient to the queue based on priority.
   ➔ Dequeue(): Removes the first patient in the queue.
   ➔ DisplayList(): Shows all patients in the queue.
   ➔ Search(): Finds how many people they need to wait in the queue.
   ➔ calculatePriority(): Determines patient priority based on symptoms.

3. **tree.h**

   Purpose: Develops a binary search tree for storing cured patients.

   Key Functions:

   ➔ createnode(): Creates a new tree node with patient data.
   ➔ insert(): Inserts a patient into the tree sorting by name.
   ➔ inorder(): Traverses the tree in-order to display patients.
   ➔ SearchTree(): Searches for a patient by name in the tree.

4. **doctor.h**

   Purpose: Develops doctor interface.

   Key Functions:

   ➔ loadFromCSV(): Loads patient queue from CSV file.

- ➔ saveToCSV(): Saves patient queue to CSV file.
- ➔ DoctorMenu(): Main doctor interface.
- ➔ MoveData(): Moves patient from queue to cured tree.
- ➔ saveCurePatient(): Saves cured patients to CSV file.
- ➔ DisplayCurePatient(): Interface of display cure patient functions.

5. **patient.h**

Purpose: Develops Patient interface.

Key Functions:

- ➔ PatientMenu(): Main patient interface.
- ➔ AddQueue(): Gets patient information and enqueues.
- ➔ SearchQ(): Check patient position in queue.
- ➔ CancelQ(): Remove patient from queue.

6. **login.h**

Purpose: Develop user authentication system.

Key Functions:

- ➔ register_patient(): Creates a new patient account.
- ➔ login_patient(): Authenticates for patient.
- ➔ is_doctor(): Authenticates for doctor.
- ➔ doctor_panel()/patient_panel(): Open doctor/patient interface.
- ➔ doctor_menu()/patient_menu(): Inteface for doctor/patient authentication.

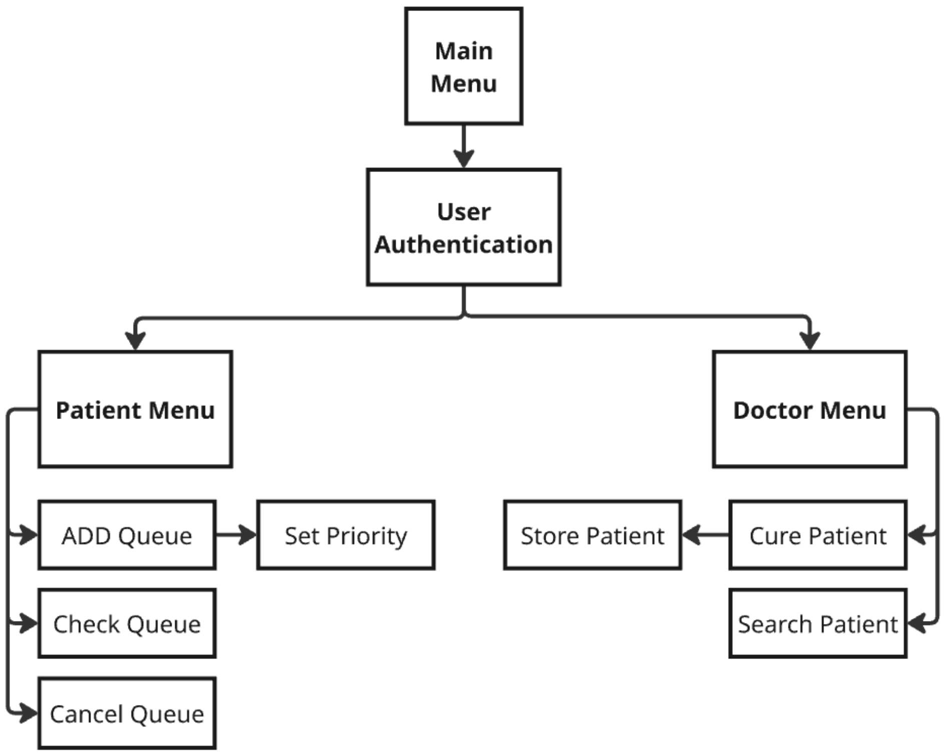
7. **main.c**

Purpose: Main running file of the program.

Key Elements:

- ➔ MainMenu(): Display main menu.
- ➔ main(): Main function.

Data Flow



Startup:

Main menu for user role selection

Patient Flow:

1. Register or login
2. Join queue, check position or cancel
3. Priority calculate base on health questions

Doctor Flow:

1. Login with password and username
2. View queue, treat patients
3. View or search cured patient record

Persistence:

1. Patient queue save to "File/patients.csv"
2. Cured patient save to "File/cure_patient.csv"
3. User account save to "File/account.csv"

Key Features

1. Priority queue show that urgent patients are seen first
2. Binary search tree for efficient storage or get information of cured patients
3. CSV base for all data
4. Input validation and error handling
5. Clean terminal interface with consistent formatting

Dependencies

The system uses standard C libraries:

1. stdio.h (file I/O, printf/scanf)
2. stdlib.h (memory allocation)
3. string.h (string operations)

Complexity Analysis

Function Name	Time Complexity
createQ()	O(1)
DeleteByName()	O(n)
Enqueue()	O(n)
Dequeue()	O(1)
Peek()	O(1)
DisplayList()	O(n)
Search()	O(n)
calculatePriority()	O(1)
createnode()	O(1)
insert()	O(h)
inorder()	O(n)
SearchTree()	O(h)

Where:

- n is the number of patients in the queue or tree.
- h is the height of the binary search tree.

Priority Queue

1. **createQ()**

   Time Complexity: O(1) because there are only a few malloc operations and strcpy operations.

2. **DeleteByName()**

   Time Complexity: O(n), where n is the number of patients in the queue. It transverse over the linked list to find the patient and delete them.

3. **Enqueue()**

   Time Complexity: O(n), where n is the number of patients already in the queue. It transverse over the linked list to find the correct position based on priority.

4. **Dequeue()**

   Time Complexity: O(1), as it simply removes the patient from the front of the queue.

5. **Peek()**

   Time Complexity: O(1), it just return the front of the queue.

6. **DisplayList()**

   Time Complexity: O(n), where n is the number of patients in the queue. It transverse over all the patients.

7. **Search()**

   Time Complexity: O(n), where n is the number of patients in the queue. It searches through the list linearly.

8. **calculatePriority()**

   Time Complexity: O(1), it just gets the user input and calculates.

Binary Search Tree

1. **createnode()**

   Time Complexity: O(1), as it only allocates memory and initializes the node with the patient's data.

2. **insert()**

   Time Complexity: O(h), where h is the height of the tree. In the worst case, this could be O(n), where n is the total number of patients.

3. **inorder()**

   Time Complexity: O(n), where n is the number of patients in the tree. It transverse to all nodes.

4. **SearchTree()**

   Time Complexity: O(h), where h is the height of the tree. In the worst case, it can be O(n) for an unbalanced tree.

Team Member Responsibilities

- Mr. Yotsapoom Liupolvanish:
    - Develop Authentication System and User Interface.
- Mr. Chaiyaphoom Chenchirotphiphat:
    - Develop Doctor Function and Tree Implementation.
- Mr. Atip Infa-udom:
    - Develop Patient Function and Priority Queue Implementation.