CS 213 Minor Project
Hospital Management System

Dhruv Ilesh Shah — 150070016
Parth Jatakia — 15D260001
Pranav Kulkarni — 15D070017
Shashwat Shukla — 150260025

April 26, 2017

Overview

Code Repository: The code for our project can be found **here**.

We have developed an interactive system for Database Management of the patients of a General Hospital. The system has all the basic functions required to manage patient records and direct all the patients to their respective doctors based on the symptoms they display and also has functionalities to perform fast and efficient searches through the database of patients. The searches can be a partial name search (if full name is not known) or full name search for finding old patient records, as well as by ID, or by the disease that said patient is suffering from. Patients records contain their entire medical treatment history and these records are stored in a central databased that can be queried. The records can also be edited.

We implemented an algorithm that directs patients to the right doctor based on symptoms, such that the patient has to wait the least amount of time. Note that a patient can have multiple symptoms and will then have to visit more than one doctor. This has also been taken into account.

Each doctor has a set of specialisations, and a queue associated with him. Patients are added to the queue of the doctor that can treat his/her symptom and the doctor's queue is also the shortest. Patients with multiple symptoms are first routed to one doctor, who after his diagnosis, directs this patient to the next doctor. We have also added an Emergency functionality, which is like the 24x7 emergency ward of the Hospital.

We have heavily used classes in our code and our implementation conforms to Object Oriented Programming Principles. Doctor, patient, diagnosis, patientrecord are all classes with their own private data members along with public accessor and mutator functions.

We have implemented and used multiple data structures to implement fast searches and to store our data efficiently. These custom-implemented data structures have also been implemented as classes. One highlight of our code is an innovative data structure that we developed, the HashMap. This functions as a 2D hash table and allows us to search to implement first name and last name search very efficiently.

Powered with all these functionalities we have made a light and colorful UI with an intuitive display of the doctor's queues, a log of what actions have performed and the status of the patients. This UI emulates the interface seen at the reception of our hospital and would prove very useful in assisting patients.

In this project apart from the custom data-structures and algorithms that we designed, we have made heavy use of things discussed in class including STL queues and vectors, Breadth First Search, tries, hashing and chaining etc.

Data Structures Explored

2D Hash Maps

While prefix tries are very efficient if we wish to search using partial queries or by first name, the prefix nature of tries means that searching via last name is not efficient using prefix trees.

We know that Hash tables provide **0(1)** access when there is low load on the table. In a typical hash table, the hash compression function maps values to a contiguous range of integers which can be interpreted as the indices in an array. We extended this idea to come up with a 2D hash table, that we call the HashMap.

The idea is that 2D arrays have elements whose location is defined by a tuple **(x,y)**. We hash the first name to find **x** and we hash the last name to find **y**. Hence searching by either first or last name can be thought of slicing a 2D array to get a row or column. A 2D array also provides **O(1)** access to all it's elements and hence, searching using both first and last name is **O(1)**. This of course comes at the cost of higher space complexity than prefix tries. Note that chaining has been implemented to handle collisions.

Tries

To implement the partial name search the best way is using a **Prefix Trie** tree. The Tries class was implemented from the scratch using nodes with functionalities of adding a patient, searching for a patient and removing a patient.

BFS has been used to search for partial matches. The search returns a vector containing pointers to the records of all patients whose name starts with the search query.

Hence the time complexity of a partial search is **O(dm)** where **d** is the length of the query and **m** is the size of the alphabet, which here is 26.

Queues

A waiting queue called patietnLine is maintained for each doctor which contains pointers to the patients assigned to each doctor. When a patient comes to the reception, based on the symptoms, he/she gets added to the waiting queue of the doctor who presently has the smallest queue. The doctor diagnoses the patient who is at the front of the queue for all the symptoms which he can cure. As each symptom is addressed, the corresponding diagnosis is pushed into the vector prescription of the patient which is the record of all the diagnosis of the patient so far. Note that a diagnosis comprises the tuple of the disease identified and the corresponding prescribed treatment. Once all the symptoms corresponding to the doctor are addressed, the patient is popped from the queue.

If the patient still has symptoms corresponding to some other symptoms, he/she is then added again to the shortest queue corresponding to these symptoms. Two important functionalities are handled here - first, balancing the waiting queue length for multiple doctors who can cure same symptoms and second, addressing multiple symptoms (corresponding to the same doctor as well as corresponding to multiple doctors).

If the patient arrives at the reception with emergency tag, the patient is added directly to the front of the queue of the corresponding doctor.

Screenshots

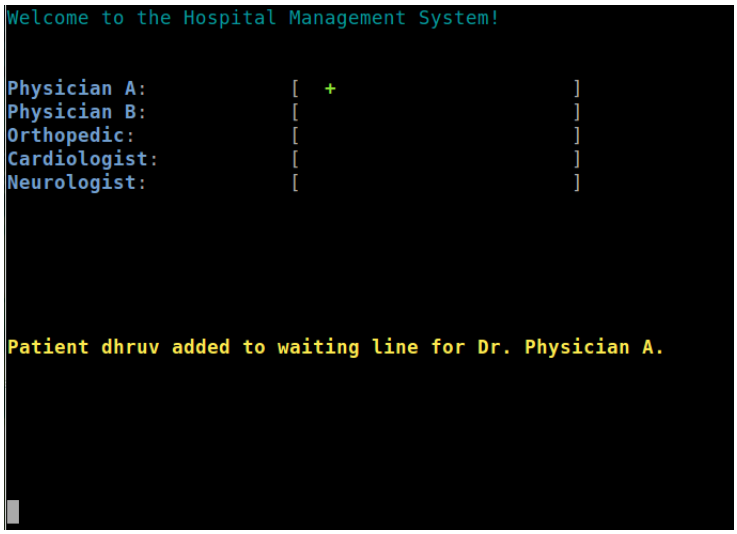

Figure 1: A patient is added to a doctor's queue

Figure 2: Queues of doctors and options available

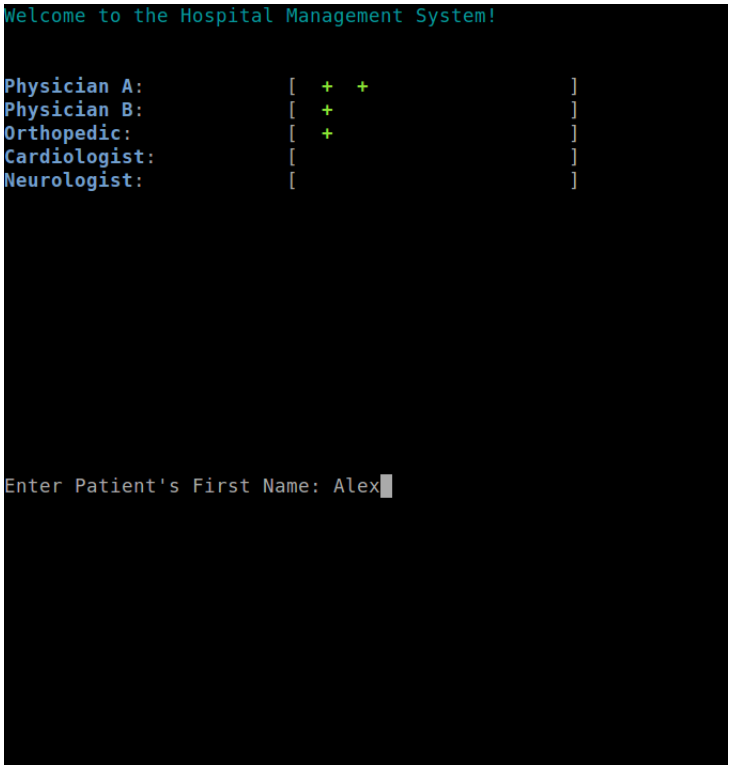

Figure 3: Adding a new patient

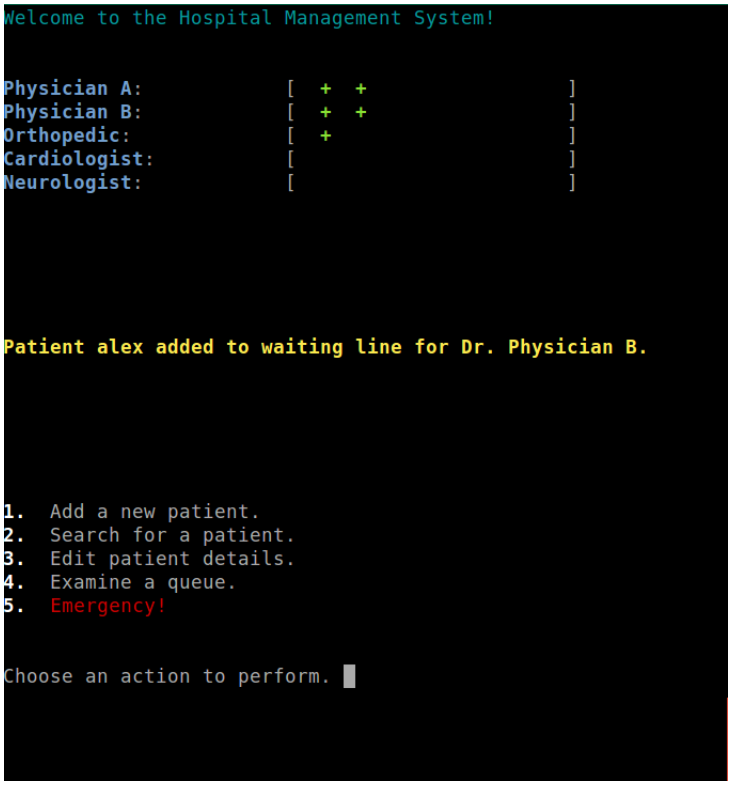

Figure 4: Currently five patients in the hospital, log of activity and available options

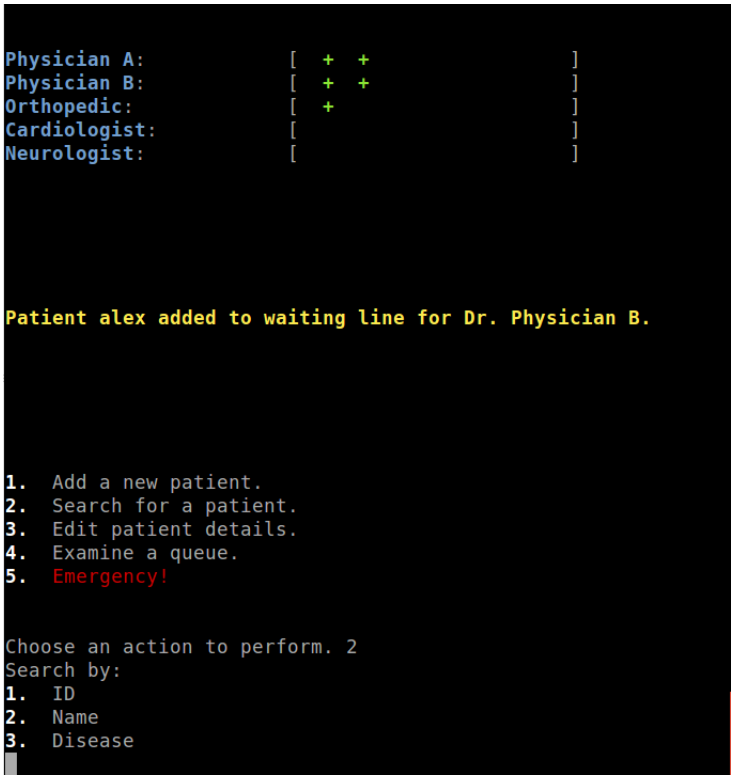

Figure 5: Searching for a patient