CS 213 Minor Project
Hospital Management System

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

April 27, 2017

Overview

1. Developed an Interactive Database Management System.
2. Functions for managing patient’s records, routing patients to their doctors, fast and efficient searches.
3. Extensively defined doctors with set of specialisations, and a queue associated with it.
4. Implemented algorithm for routing Patients to shortest Queue.
5. **Emergency** functionality to emulate the Emergency Ward.
6. Quick Searches of full names and even partial names.
7. 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.

Data Structures Explored

2D Hash Map

Efficient search by name

At the heart of the hospital database is a hash table that performs unique hashing by two keys (fname, lname). For collision handling, chaining has been implemented in the class *HashMap_2D*.

- ➔ *insert_patient (patient * pat)*: Generate the 2 hashes and append the patient into the hash node, accounting for collision and chaining.
- ➔ *search_patient (string fname, string lname, vector<string> &IDs, vector<patient *> &patients)*: Search in the hash table for a patient with given fname, lname. ***O(1)*** *access and return.* The function returns a vector of patients and their IDs for convenience.
- ➔ *Search_patient_lname, search_patient_fname:* Similar to the above function, but for partial searches. ***O(t)*** *access and return, where* ***t*** *isTABLE_SIZE.*

The complete search can be performed pretty efficiently using the *HashMap_2D*.

Tries

Partial String Search

To search for a Patient with only beginning partial part of name known Tries was implemented. The *class Tries* has three important functions:

➔ *insertPatient (patient* newpatient)* - To add new patient to the Trie.

➔ *searchPatient(string name)* - To search for patients with the matching string 'name' in their beginning part of their name, by performing a BFS on the trie.

➔ *removePatient(patient* Rpatient)* - To remove a patient from the Trie

Due to *Tries* searching partial names can be implemented in ***O(dm)***.

Queues

Routing of Patients

At the receptions , based on the symptoms, patients gets added to the waiting queue of the relevant doctor who presently has the smallest queue. Following are the important functions:

➔ *assignDoc(patient * p)* - Finds the doctor corresponding to the symptom and adds the patient to the waiting line with minimum waiting queue using function *findMyDoc(patient * p)*

➔ *doctor::diagnose()* - Doctor examines patient at the front of the waiting queue for all the symptoms that the doctor can cure. Symptoms are cured, popped and the corresponding diagnosis is added into the prescription/logs of the patient. If all the symptoms are not addressed, patient is added to waiting line of another relevant doctor.

➔ *Emergency (patient * p)* - The patient is added directly to the front of the queue of the corresponding doctor

Miscellaneous

Challenges Faced

- Integrating several independent components into a complete package.
- Creating a simple, yet exhaustive, UI to visualise, debug and present the underlying data structures implemented.
- Managing our code across 3 different operating systems and between shell & IDEs.

Future Prospects

- ➔ To maintain blood group directory by storing blood group of each patient and adding functionality of search by blood group.
- ➔ Search patient by partial string from name, not necessarily form the start - Using wild char in the string search by tries.
- ➔ Skip lists for faster partial search.
- ➔ A suffix trie with smart BFS can be implemented for wildcard matching and better partial search.