1. Introduction

This is the Computing Assignment 1 of Group E4. In this assignment, we handle and implement a registration, queueing and reporting system for medical treatment. Later, we will introduce how we implement the codes in details and how to check our codes performing correctly.

Appointment_hospital.cpp, Create_new_file.cpp, Fb_heap.cpp, local_queue.cpp, report.cpp,reportlist.cpp are files containing classes. Head.h is the head file. Test.cpp is the main function. Two test input data files are already given. You can sue create_new_file.cpp to

2. Main structure

Our model simulate three month's medical treatment system. In our main program, there's 5 time relevant parameters, namely, month, week, day, half day, and hour, each start with zero, and iterates as program processes. There are also parameters of three local queues, three hospitals , one fibonacci heap, and one report list. See implementation part for these class's details.

We use six numbers to 0 to 5 to mark the status of one person. Zero means a person is newly registered and is still in local queue. One means a person is in the heap. Two means that person has appointment assigned but the appointment time has not passed. Three means that person's appointment time has passed. And four means that person has not come to the treatment, a person can only reach this state when he withdraws from state three. State five is marked when a person withdraws when he is in the heap or waiting for appointment, that is, when he withdraws from state one and state two.

Our main program reads line by line from the input file, and if this line is a line of a person's information, we store these information into that person's class and push that person into the corresponding local queue.

If that line is an end mark that marks the end of half day, it means half day has passed and we pop all the people from each local queue. If this person is newly registered, we simply insert him into the fibonacci heap and report list and mark him as state one. If that person is already in the report list, and has state one, that is in the fibonacci heap, we update and reorganize the fibonacci heap if that new line of person's information is update, and kick him from the fibonacci heap if that line of information is withdraw and state is changed to be five. If that person is already in the report list and has state two, we update the information if this is an information update and remove that person from the appointment list if this is a withdraw and state is changed to be five. If this person is already in the report list and has state three, we update the information in the report list if this is an update and mark that person do not come to treatment as state four if this is an withdraw. If that person is already in the report list and has state five, this means that person registers again after withdraw, and we punish him accordingly and insert him to fibonacci heap, his state also changes back to state one.

Every half daily, we also check all the person in the report list with state two to see if their appointment time has passed and mark them as state three if so. Every day, we pop half of the people from the fibonacci heap with minimum key values and assign them appointments and changes their state to state two. Every week and every month, we generate the weekly and monthly report.

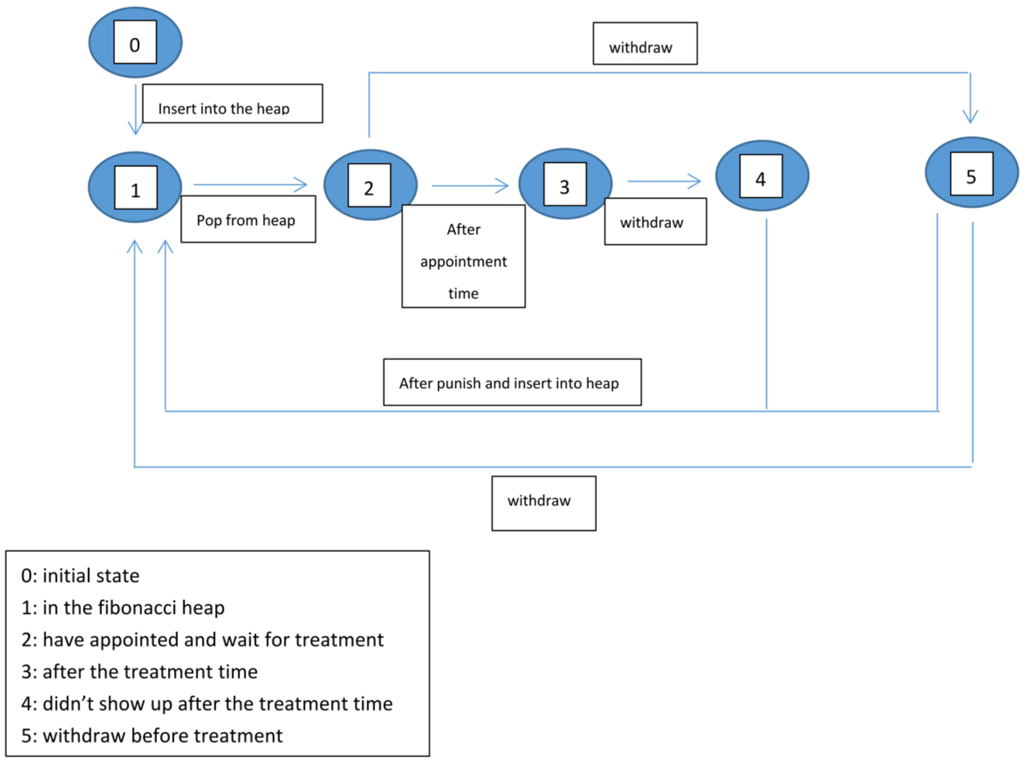
0
Insert into the heap
1
Pop from heap
2
After appointment time
3
withdraw
4
5
withdraw
After punish and insert into heap
withdraw
0: initial state
1: in the fibonacci heap
2: have appointed and wait for treatment
3: after the treatment time
4: didn't show up after the treatment time
5: withdraw before treatment

Figure1. State Transformation

3. Implementation

3.1 Registration Record and Local Queue

We design three classes: *person*, *fifo*, *TableWrite* to create a file containing all people who have registered.

Person: It contains all of the information of a person including id, name, WeChat, whether this person has withdrawn or not etc. All the information is crated randomly using the function *random_generate()*. To make the information more realistic, we set addresses into four real places in Haining. For simplicity, we assume each month has 28 days and the minimum unit of time we use is 0.1 hour and we simulate for three months. The corresponding time in reality is from 2022-1-1 00:00 to 2022-3-28 23:54. Based on the information, *set_key()* calculated the priority of this person to meet the requirement of the priority addressing. One constructor of person also takes a table line as input so when a table line is read from input file, a person can be created with the information of this line.

fifo: the First-in First-out Queue. It is the data structure for local queue. All people in one local registry are stored in one fifo type local registry.

TableWrite: The class that creates a file containing all people who have registered. The file contains ID, Name, Address, Phone, WeChat, Email, Profession, Age, Risk Status, Registration Time, whether this person is going to withdraw, whether this person has presented a priority letter, which local registry this person is in and the hospital ranking list of this person. Every half day is passed, the mark “end of half day” is created since we process patients every half day. The file created may have some displaying problem due to the displaying way of csv, but the data has no problem and you can change the setting of csv document or click the item to completely display the data.

TableWrite’s constructor takes the argument as follows: (filename, num, updated_num, withdraw_num). num is the total number of people in the file. Updated_num is the number of people who update their information. Withdraw_num is the number of people who withdrew. Update is represented with two table lines. These two table lines represent the the same person but their attributes may be different. The second line is located in the half day on which this person’s information is updated. Withdraw is represented with two table lines as well. These two table lines represent the same person but the registry time is different. In addition, the latter line is marked with withdraw item being 1. Once a variable of TableWrite type is created, one can use table_create() function to generate new registry data files.

To **create new test files**, you can create TableWrite object, set the parameters as above and call table_create() function.

3.2 Fibonacci Heap for the Central Queue

As for the correct implementation of the fibonacci heap and priority rule, when writing the codes, we refer to the description of the operations insert, delete min, decrease, consolidate and so on in the ppt of lecture. And also add some new operations like search particular node in fibonacci heap and update(choose increase or decrease key value), referring the description in website blog. To test the correct implementation of fibonacci heap and priority rule, you can just test by using insert and delete_min operation and determine whether the key value of the nodes popping from the heap is in the order from small one to the larger one. So I write one function called showpriority() in which it delete_min for all nodes in fibonacci heap, print out person’s id and key value and then insert the person back to heap, which can show the correct implement of fibonacci heap and priority. If you enter 1 to see the demo of withdraw and update, you can also see that the order in fibonacci heap is correct as the fibonacci heap pop out according to their key values, that is from the smallest to biggest.

As for the order from local queues is preserved, in our codes, we insert every person in local queue in order. When they are inserted into the central fibonacci heap, they are sorted only corresponding to the priority key value. As in our algorithm, key value is calculated by time, age, profession, high risk level, priority letter with the weight, so it is impossible to compute same key value in our code. So we need not consider the condition when two people with the same key value. So that all the order from local queues is preserved only corresponding to different key value.

As for the function that the registration's attributes can be updated, and the priority is correctly reflected, in our codes, we have functions update(newpatient, oldpatient). When we need update the attributes of one person, we only first determine whether this person is in the repor t list. If find this person in the report list, then check his status, if he is in the state 1, which means he is still in heap, we use the update function to replace the person's old information with the new information and reorganize the heap's structure. If you enter 1 to see the demo of withdraw and update, you can see how the fibonacci heap's member's priority order changes after one person in the heap's information is updated.(ie. Person with id 0899036388 writes a priority letter, and his order in the fibonacci heap changes from the 4th to the 1st)

As for the withdraw when that person is in the fibonacci heap, checking that that person's status is in the heap, and he withdraws, we kick that person from the fibonacci heap, and mark that person's status to be state 5, and next time when we see that a person with state 5 in the report list register again, we use person's member function punish() to add two weeks to his register time for punishment unless he has high or medium risk, and recompute his key value, than we insert him as usual into the fibonacci heap. You can enter 1 at the beginning to test this functionality.

3.3 Hospital and Appointment

After a person, as the min in the present heap, is popped out of the fibonacci heap, we will set up a class object "appointment" for that person, and use a member function make_appointment() to find the appropriate appointment time and place. That person will be marked with state 2 which means he has an appointment but the treatment time has not passed. To be specific, the default time for appointment is the next day after he is out of fibonacci heap, and we check the person's preference list for hospitals, loop through that preference list until we find one available hospital for that person, if no available hospital is found for next day, we increment the date and repeat the procedures above. We also implement a class called "hospital", which is assigned with daily capacity, opening time, and close time. This "hospital"class can use these data to check whether there's still time slot for treatment at that day, and if there is, the hospital can give suitable treatment time slot for that person. When an appointment is successfully made, we add the pointer of that appointment to a vector list in a hospital class object. Each hospital has such a list that stores appointment information.

If a person withdraws after an appointment is made but before the treatment time, there's a withdraw_app function to remove that person from the corresponding appointment list and set that person's appointment pointer back to null. We also have a is_appointment passed function that checks all the people with state 2 to see if his appointment time has passed, if so, change his state to be state 3. If we later find that this people have a withdraw mark in the input file, it represents that this person do not come to the treatment, we mark that person’s status to be state 4 in the report list. These status marks will also be used when generating reports. You can enter 1 at the beginning to see how the appointment list changes when one person withdraws.

3.4 Report

To generate weekly reports and monthly reports, we create a vector list called “report list” which in pushed into all the information about that person and then we write the information which is from the report list into weekly reports and monthly reports.

For weekly reports:

Firstly, we need to input the information including their profession category, age category, risk status and the waiting time of the three types of people: the people who have been treated, the registered people with a set appointment and the queueing people without a set appointment. So, by checking the patients’ status in report list, we produce three files per week, and each file includes one type people, i.e., “Weekly1File1” for people without appointment(status==1) in the 1st week; “Weekly3File2” for the registered people with a set appointment(status==2) in the 3rd week; “Weekly7File3” for treated people(status==3) in the 7th week. Because our test time is 3-months, there are totally 36 files are created.

Secondly, to order these reports by name, profession category or age group, we create a sort function, by which a user is provided an input to choose the way he need to order the report. In this function, we compare each parameter in the report list and arrange them in ascending order, and then the other 36 sorted files are produced as well. The file name is in “SortedWeekly<m>File<n>” format, i.e., “SortedWeekly1File1” means this file is the sorted files for people without appointment(status==1) in the 1st week.

In addition, to make the files well-formatted, we choose to generate “.csv” files which can be open in Excel format.

For monthly reports:

We write a function to put in the following information which is got from the report list into monthly reports: how many people have registered, how many of them are waiting, how many are waiting in total, how many treatment appointments have been made, the average waiting time, and the number of people who withdrew their registration. Because the test time is 3-months, there will be three monthly reports generated, and the file names are “monthly1_report.txt”, “monthly2_report.txt”, “monthly3_report.txt”.

4. Instruction

1.First type “make” in the terminal to compile our program.

2. **IMPORTANT:**

Enter 1 to test a small set of data (only two day’s register information, no weekly and monthly report).

Enter any other number to test a big set of data of three months.

3. Enter 1 or 2 or 3 to generate sorted weekly report. There are 12 weekly reports in total, so you need to enter these numbers for 12 times. Please be patient :)