Lab: Staffing Problem

CSI 3305: Introduction to Computational Thinking

December 13, 2010

1 Introduction

Although certain images come to mind when one thinks of a nurse, the majority of those are not reflective of what the actual job involves. Although health care today requires a team approach, nursing is an autonomous profession that requires independent thinking and split second decision making. Nursing is a job that is rigorous and fast paced. On a daily basis, nurses make over 100 decisions that affect a persons health and in many instances, their ability to maintain life and well being. Decisions must be made in a variety of areas including staffing of patient care based on acuity and prioritization, interactions with the multidisciplinary health care team, patients and families, and safety of the environment.

One of the major reasons nurses report for leaving the profession is their inability to give the level of patient care they believe is best due to inappropriate workloads. Staffing decisions and patient assignments are often based on a patient acuity system, which is not well defined and receives little nursing input in its development.

2 Problem Statement

Your task is to code a strategy for assigning nurses to take care of patients, and a hospital to simulate your strategy.

We will explore 2 strategies for assigning nurses:

1. Assign as many nurses as needed to get a patient healthy according to which room the patient is in
2. Assign nurses to patients in critical condition first before assigning patients to healthier patients

3 Tools

You will be using the Python programming language and Python interpreter for this lab. The code will be written using a text editor, and the interpreter will be called using the command line.

4 Setup and Programming

4.1 Python

1. To begin, create a new text file and label it staffing.py.
2. Open the text file in the text editor of your choice (such as Notepad or Vim.)
3. Enter the following code at the top of your file:

   ```
   from __future__ import division
   import random
   import math
   ```

   (This code will allow Python to handle division in a more intuitive way, doing floating point division by default, and some additional libraries)
4. Next, we need to define some constants that will be used throughout the program. Type the following code into your file:

   ```
   TOTAL_NURSES = 15
   NURSING_ABILITY = 10
   ```

   TOTAL_NURSES is total number of nurses we have in the hospital, and NURSING_ABILITY is just a measure of how much a nurse can heal a patient.
5. Next, we will create some functions. Enter the following code, beneath what you've already entered:

   ```
   def nextDay( patients, nurses ):
       nlist = []
       for i in range(len(patients)):
           if patients[i] <= 0 or patients[i] >= 100 :
               nlist.append( 90 )
           else:
               nlist.append( patients[i] - random.randint(0, 30) + nurses[i]*NURSING_ABILITY )
       return nlist
   ```

   This first function simulates a day in the hospital. We go through the list of patients and one of two things happen:

   (a) The patient died (patient's health reaches 0) or the patient completely recovered (patient's health reaches 100), so we remove the patient and admit a new patient with a health of 90, OR

   (b) We simulate the patient recovering, first we subtract a random amount between 0 and 30 to simulate the patient getting worse, then we add the amount of health which the nurses restore.

   Next, we add a function to check if any patients died:

   ```
   def checkAssignment( patients, diedPatients, recoveredPatients ):
       for patient in patients:
           if patient <= 0:
               print "a patient died..."
               diedPatients += 1
           elif patient >= 100:
               print "a patient has been discharged!"
               recoveredPatients += 1
       return diedPatients, recoveredPatients
   ```

Notice that this is very similar to part of nextDay(). Again, we go through the list of patients, and if any of the patient's health reaches or drops below 0, we report that the patient died, and add the patient to a diedPatients counter. On the other hand, if any of the patient's health reaches or goes over 100, we report that the patient has recovered and discharged, and add the patient to a recoveredPatients counter.

6. Here we actually implement our strategy of assigning nurses:

```
def assignNurses( patients ):
    nurses = [];
    availableNurses = TOTAL_NURSES
    for patient in patients:
        nursesNeeded = math.ceil((100-patient)/NURSING_ABILITY)
        if availableNurses > nursesNeeded:
            nurses.append( nursesNeeded )
            availableNurses -= nursesNeeded
        else:
            nurses.append( 0 );
    return nurses;
```

This is a very simple strategy. We go through our list of patients, and for every patient we calculate how many nurses a patients need to fully recover, if we have enough nurses we assign that number of nurses to the patient. Otherwise if we don't have enough nurses, we don't assign any. Notice that we go through the patients in the order of which room they are in (their index in the list).

7. Lastly, we enter the code that simulates the hospital:

```
''' Variables '''
patients = [90,90,90,90,90,90,90,90,90,90]
currentDay = 1
totalDays = 100
diedPatients = 0
recoveredPatients = 0


''' Simulate the hospital '''
while currentDay <= totalDays:
    print "Day", currentDay
    print "patients:", patients
    nurses = assignNursesBetter(patients)
    print "nurses:", nurses;

    currentDay += 1
    patients = nextDay(patients, nurses)
    diedPatients, recoveredPatients = checkAssignment( patients,
        diedPatients, recoveredPatients)
    print ""

print "after", totalDays, "days,", diedPatients, "patients died and ", \
        recoveredPatients, "were discharged."
```

First we define a few variables:

(a) The initial list of patients

(b) The starting day

(c) How many days to simulate

(d) How many patients died

(e) How many patients recovered

The while loop just simulates the days until we reach the amount we specified. At the beginning of each day, we first print out the day and the health of the patients. Then we assign nurses to the patients, and print out the nurses. We then simulate the day, so the day passed and we increment currentDay, and we call nextDay to simulate the patients recovering. At the end of the day, we check to see how our patients did.

After the while loop completes, we print out how well our strategy did.

8. Save the file and open the command line (From the Windows menu bar: Start > Run... > cmd) Navigate to the folder where you created your file (use the following command: cd "C:\Folder\where file\is".)

9. From the command line, type the following to run your program and hit enter:

```
python staffing.py
```

10. Notice that this strategy does not do very well as quite a few patients die. This is because by the time we reach the last room, there is a good chance that we have already assigned all the nurses jobs. Instead, let's consider a better method. A better approach would be to assign nurses to the sicker patients (health nearer to 0) first. This way we can prevent the sicker patients from dying.

    Type this code below assignNurses().

```
''' Assign nurses to patients in a smarter manner'''
def assignNursesBetter( patients ):
    sortedPatients = sorted(patients)
    patientsCopy = list(patients)
    sortedIndex = []

    for patient1 in sortedPatients:
        for index, patient2 in enumerate(patientsCopy):
            if patient1 == patient2:
                sortedIndex.append( index )
                patientsCopy.pop( index )
                patientsCopy.insert( index, 999 )
                break

    nurses = [0,0,0,0,0,0,0,0,0,0];
    availableNurses = TOTAL_NURSES

    for index in sortedIndex:
        nursesNeeded = math.ceil((100-patients[index])/NURSING_ABILITY)
        if availableNurses >= nursesNeeded:
            nurses.pop(index)
            nurses.insert(index, nursesNeeded)
            availableNurses -= nursesNeeded
    else:
```

```
                nurses.pop(index)
                nurses.insert(index, 0)

        return nurses;
```

In this code, first we sort the patients by health into a new list called sortedPatients. Then we make a copy of the patients for later use and create a sortedIndex list.

Although we have sorted the patients, we can't just assign nurses using this list, because the room numbers are no longer correct. To fix this, we use our sortedIndex list. We go through our list of sorted patients, then for each of the patients in this sorted list, we find the room that the patient is in by comparing the health of the patient in the sorted list (patient1), and the health of the patient in the current room (patient2). Once we find the room the patient is in, we add the room number to the sortedIndex list. We then remove the patient and replace it with a value that cannot be any patient's health.

After the last step, we now have a list of rooms of the sickest to the healthiest patients. We initialize a nurses list because this time we can't just append to the list since we have to insert it according to patient's room number which is not in order.

We go through the list of rooms, and similar to our last strategy, we find out how many nurses are needed to nurse the patient back to full health and assign it. The only difference is that we do pop() and insert() to put the nurses in the correct rooms.

Finally, we change the function call in the hospital to call this function.

```
nurses = assignNursesBetter(patients)
```

Notice now, that we have far fewer patients dying, but at the cost of discharging less patients.

5 Questions

1. Run each strategy for 1000 days and report how many patients died and how many patients were discharged.
2. Which strategy is better and why?
3. What is another strategy that might work better?