Data Structures and Algorithms
The LNMIIT, Jaipur
Project 2 -- 20 April 2020

Due date: 09 May 2020 **Marks: 04%**

We want to study the gains of the social distancing through a simulation study. At the end of the exercise project teams will submit four tables to report their findings. The table below provides the recommended structure for each report table.

Description of experiment	Value
Population of the town (P)	
Count of major service providers (S)	
Count of known infected persons (IR)	
Result	Count of person who need to be tested -- Average of 10 runs (with standard deviation)
Without any social distancing (T100)	
With social distancing reducing contacts to 50% (T50)	
With social distancing reducing contacts to 33% (T33)	
With social distancing reducing contacts to 50% with assigned service provider (TC)	

The problem description below will help the students understand the data they must report in these tables and it also explains the computational details to compute the necessary simulation results.

A town has population (P). Three different population levels are established for this purpose. The proposed population sizes for reporting are 2000, 5000, 10000.

The people in a town are served by their service providers (Grocers, Hairdressers, Restaurants etc). These providers are part of the town population. Two separate levels of service providers are to be used in this study. The first level assumes that the town has very few providers – that is, number of service providers is 3% of the town population. At the other end is a town that has numerous service-providers; in this later case, assume the number of service providers in the town to be 8% of the population.

The final parameter of interest in this study is the count of known infected persons (IR) in the town. Three settings for reporting are: 5, 30, and 100.

Each project team needs to perform their simulation study and report 4 tables. In choosing the tables to report make sure that you have at least one table for each suggested level for each parameter discussed above. That is, there should be at least one reported table for P value 2000, 5000 and 10000. Similarly, for the other two parameters S and IR.

You are required to run each experiment defined by a set values of three parameters multiple times. Specifically, you are asked to run each experiment 10 times and report the average observed values. You are also required to compute the standard deviation for these 10 observed

simulation results. Larger values for the standard deviation indicate that the simulation result may be significantly different from the averaged value in an actual real-world occurrence of the epidemic.

The following is overview of the planned simulation. We make a closed world assumption for a town. We assume a period of a week in which people in the town interacted with one another. At the end of the week certain number of residents are found infected by a disease. Based on their recorded interactions during the week and the individuals found infected we wish to determine people who should be tested for the possible infection.

The purpose is to seek a good way to lower the contacts so that number of the vulnerable persons is small.

Creating the town population and interactions among the population over a week

For this simulation we will make several quite unrealistic but simple assumptions about the human social behaviours. The assumptions are listed below:

1. Each person interacts with about 20 other persons in a normal week. This count does not include interactions with the service providers.
2. Each service provider comes in contact with about 5% of the town population if the town has low number of providers. In town with many service providers, each provider only interacts with 3% of the town population in a week.
3. We will assume that these person-to-person and provider-to-person are biased in some ways. We will suggest a way to create this bias.

Next, we explain how students can create an adjacency matrix for the town population and enter person-to-person interactions among the inhabitants of the town. We explain this assuming a fixed population P, service level S, and infection rate IR.

First, create function `int randomYes()` that returns 1 only 0.1% of the time – 1 in 1000 calls. It returns 0, for 99.9% of the calls to the function. You should be able to write this function using code from the musical chair training set of DSA Lab. Please note that this function will be used in your program that will run 10 repeats of a single experiment in a single run of the program.

As a hint, your function `int main()` will look somewhat as follows:

```
int main(void) {
     int repeats;
     int results[10];

     for (repeats = 0; repeats < 10; repeats++)
          results[repeats] = doExperiment();

     // Compute average and std dev.
}
```

Function `doExperiment()` may use function `randomYes()` to set up the adjacency matrix at the start of each experiment. The result of the experiment will be recorded for computing values to be reported.

It should be obvious to each project team that function `doExoeriment()` performs the following actions:

1. Initialise a p*p matrix. And, use function `randomYes()` to randomly mark the required number of nodes in the matrix as service providers. The service provider count will be determined by the population size and level of service providers parameter set for all experiments in a program run.
2. To initialise each row in the matrix, repeatedly use function `randomYes()` to attempt to set entries under various columns (x) in the row (y) to 1. The number of entries in a row that need be set to 1 depends on the nature of the node associated with the row y: that is, the count is determined if the row is a service provider or a non-service provider. You are also advised to take note of the following concerns:
    a. Interaction between persons is bi-directional. So be sure to update both entries (x, y) and (y, x) in the adjacency matrix.
    b. When inserting interactions in row y, you would note that the entries under columns 0 to y-1 have already been made. Therefore, the new interaction entries are only needed for the columns y to p-1.
    c. Since, function `randomYes()` returns 1 only 0.1% of the times, you may need to repeatedly work over the row till enough entries in the row have been set to value 1.
    d. The interactions among people are rarely totally random. There is a significant bias towards interaction among the friends and neighbours. To model this consider using a biased random function. One such function `int biasedYes()` is described below. Use of this function would require you to modify the approach described above appropriately.
3. Finally mark applicable number of persons as infected in the town by using function `randomYes()`.

Once, you have set the adjacency matrix for an experiment you have a record of interactions among the population of the town over the previous one week. And, the infected persons are known. You must not forget that service providers are also included in the adjacency matrix and their interaction must be set correctly before you progress to the next stage of computation.

You can create the transitive closure of the graph. The persons who are adjacent to an infected person in this transitive closure need to be tested for any possible infection. The number of persons to be tested is returned as return value by call `doExperiment()`.

Function biasedYes()

Interactions among the people are usually strongly biased. Therefore, we suggest that teams use a function derived from `randomYes()` to create interactions between persons. To create interactions using this function the frequency of yes from function `randomYes()` has been set very low.

```
Int biasedYes(int x, int y) {
     /* Where x and y and indices of persons in
          Adjacency matrix. */

     Int common = countPrimeFactors(x+1001, y+1001);
     /* Define function countPrimeFactors() that computes the
          Count of prime factors common in two arguments.

          More prime factor they have in common, more
          often the pair meets!
     */

     // Return biased interaction
     While (common--)
          If (randomYes() == 1)
               Return 1;
     Return 0;
}
```

After you have set up the adjacency matrix, I suggest that you count the number of interactions in it. If the total number differs by more than 10% from the target, the matrix entries may need to be adjusted. [The target number of 1s in adjacency matrix is 20*p*(1-S)]. Do a random scan over the adjacency matrix to either add more interactions (randomly turn some 0 to 1) using (a variant of) `randomYes()` or to randomly remove some interactions by converting some 1s to 0s.

If it is appropriate, you may try to use function biasedYes() to rapidly increase number of person-to-person interactions in the adjacency matrix. Alternately, if there are too many 1s in the adjacency matrix, the number can be reduced by a simple walk over the elements in the matrix and converting every 1 to 0 based on a proportion determined by the needed reduction in the count.

Various Social Distancing Scenarios

Once again refer to the table that lists the various cases of restricted interactions among the populations. These restricted cases are explained below.

Without any social distancing (T100)

This case is business as usual and has already been descried in the previous section.

With social distancing reducing contacts to 50% (T50)

The person-to-person interaction is reduced to one half the level used in T100 experimentations. However, the interaction with the service providers remains at the level used in the business as usual level.

You may achieve this by performing reset of certain set values in the adjacency matrix.

With social distancing reducing contacts to 33% (T33)

The person-to-person interaction is reduced to a third of the level used in T100 experimentations. However, the interaction with the service providers remains at the level used in the business as usual level.

With social distancing reducing contacts to 50% with assigned service provider (TC)

The person-to-person interaction is reduced to one half the level used in T100 experimentations. Further, the odd numbered service provider will only interact with the persons with odd indices. Similarly, an even indexed person only gets service from a service provider with even index.

Transitive closure of adjacency Matrix

Please see Figure 13.7 in Reema Thareja's book for DSA. You may also see Lecture recording for Section A2 dated 17 April 2020 on DSA Theory classroom.google.com site.