Biostat 625 Homework #2

Submit as a compressed file "hw2.tar.gz" containing the required computer code for the following problems. Note that:

- Follow exactly the requirements stated in the problems, as your code will be tested automatically by another program, which cannot accommodate human errors such as typos in file or function names. In particular, do not include any unnecessary test code or printout.
- Consider all possible test cases and make your code as robust as you can, as all potential input arguments are allowed as long as they are not in contradiction to the problem description.
- Make your code as efficient as you can, as some of the test cases may be computationally challenging and your code will be terminated if it does not finish after running for 10 seconds and you will lose the points for those test cases. It will also be terminated if it uses more than 1GB of memory. However, you should always turn in your code even if it is not very efficient so that you can at least get partial credits.
- All your code will be tested on the biostat cluster.

Problem 1 - Find triangle

Write an R function named `triangle`, which takes an argument of a $K \times 2$ matrix named `edges`, and returns whether an undirected graph with all the edges specified in `edges` (each row of `edges` specifies the indices of the two vertices in the graph that the edge connects) has any triangle (i.e., any cycle with length 3) in it. Save your function in a file named `triangle.R`. Example runs are given below.

```
> triangle(matrix(c(1, 2, 2, 3), 2, 2))
[1] FALSE
> triangle(matrix(c(1, 2, 3, 2, 3, 1), 3, 2))
[1] TRUE
```

Hint: matrix and vector operations are more efficient in R.

Problem 2 - Matrix Kendall

Given observations $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, the Kendall's τ, a non-parametric rank correlation coefficient, is defined as

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{n(n-1)/2}$$

where for any $1 \le i < j \le n$, the pair of observations (x_i, y_i) and (x_j, y_j) is called concordant if both $x_i > x_j$ and $y_i > y_j$ or if both $x_i < x_j$ and $y_i < y_j$, and discordant otherwise, i.e., if $x_i > x_j$ and $y_i < y_j$, or if $x_i < x_j$ and $y_i > y_j$. Here we assume all the x's are different, as are all the y's.

Write an R function named `matrix_kendall`, which takes an argument of a $m \times n$ matrix named $\mathbf{X}$, computes the Kendall's tau statistics between each row of $\mathbf{X}$ and $\mathbf{y} = (1, \ldots, n)$, and returns the maximum value of τ achieved across all the rows of x. Save your function in a file named `matrix_kendall.R`. Example runs are given below.

```
> matrix_kendall(matrix(c(1, 2, 2, 1), 2, 2))
[1] 1
> matrix_kendall(matrix(c(2, 2, 1, 1), 2, 2))
[1] -1
```

Hint: optimize your code, as some of the test cases may be challenging.

Problem 3 - Number string

Define S as an infinite string formed by concatenating all the natural numbers in increasing order, i.e.,

$$S = \text{“}12345678910111213141516171819202 1 \ldots 99100101 \ldots \text{”}.$$

Write an R function named `num_str`, which takes an argument of a sequence of digits named `s`, returns the first location in S that `s` appears. Save your function in a file named `num_str.R`. Example runs are given below.

```
> num_str("12")
[1] 1
> num_str("21")
[1] 15
```

Hint: optimize your code, as some of the test cases may be challenging. Also, modular design can help avoid bugs.

Problem 4 - Elevator

A building has levels numbered as $1, 2, \ldots$. It has an elevators in it (with infinite capacity). At time $t = 0$, the elevators is resting at level 1. There are n requests $x_1, \ldots, x_n$. Each x_i is a triple $x_i = (t_i, s_i, d_i)$, which means that a guest made a request at time $t_i > 0$ at level s_i, and the guest is going to level d_i, where $s_i \neq d_i$. The elevator operates as follows:

1. It will keep going in one direction (i.e., up or down) until there is no guest in it, and there is no other guest waiting to be served at a further level in the current direction.

2. It will collect guests that are going in the same direction along the way.

3. After it stopped going in one direction, it will turn around and go in the opposite direction, if there are any pending requests from guests.

4. It takes one unit time to move up or down a level, and takes one unit time to stop at a level for all the guests to enter and/or exit (if any). Guests arriving during the door opening period are still able to get in.

5. For scenarios that depend on the exact equality of two events, e.g., two guests making requests at exactly the same time both above and below a stopped elevator, or a guest making request at exactly the same time when the elevator passing by his/her level, the elevator can take either reasonable action. In fact, such scenarios will not occur in the test cases.

Write an R function named `elevator`, which takes an argument of a $n \times 3$ matrix named `x` which contains all the requests as described above, and returns the time when the last guest arrives his/her destination. Save your function in a file named `elevator.R`. Example runs are given below.

```
> elevator(matrix(c(1, 1, 2), 1, 3))
[1] 4
> elevator(matrix(c(1, 1.5, 2, 3, 3, 2), 2, 3))
[1] 7
```

Hint: modular design can help avoid bugs.