EEL 6761: Cloud Comp Sys. And Appls Spring 2024 Instructor: J. Fortes

Homework 2 Assigned: 1/11/2024 Due: 2/13/2024 @ 5 Pm (use UF E-learning - Canvas)

Topology Management in Peer-to-Peer Systems

Preamble:

In this homework, you are required to write a C++/Java/Python program to implement the method for "Topology Management of Overlay Networks" described in Section 2.2.2 of Andrew S. Tanenbaum's Distributed Systems book and in attached reading supplement 1 (The paper "T-Man: Fast Gossip-based Constructions of Large-Scale Overlay Topologies" by Mark Jelasity and Ozalp Babaoglu). This algorithm is also known as Jelasity and Babaoglu's algorithm. The gist of this algorithm is discussed in the next paragraph (use the above references for the detailed descriptions).

In this algorithm, every node in the network maintains a list of neighbors. During the network-initialization phase, each node randomly selects k neighbors and places them into its neighbor list. During the network-evolution phase, in each cycle of the iterative algorithm, every node randomly selects one of its neighbors, and then sends a list consisting of the identifiers of its neighbors and of itself to that neighbor. The selected neighbor also sends its neighbors list back to the node which initiated the action. Upon receiving the new neighbor list, the nodes select the nearest k nodes from both the new and old lists as their neighbors and discards all the others. The definition of distance could vary depending on the application. For instance, in CHORD, distance is defined in terms of the hashed value of the node ID. In other applications like storing music album information, distance could be defined in terms of the genre of music. Thus, nearest nodes would contain information on either the same genre of music or similar genre of music.

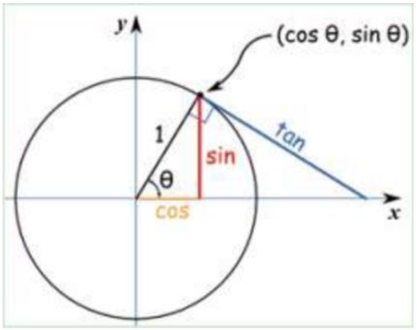
You will write a sequential program that implements Jelasity and Babaoglu's algorithm so that in each cycle the nodes in the network initiate communication with each of their neighbors one by one. Your program MUST accept N, the total number of nodes in the network, and k, the number of neighbors each node maintains as the input parameters.

Tasks:

There are 2 tasks as part of this homework which are elaborated below:

1. (**Ring topology**): Consider the case where the nodes are identified using colors represented in (Red, Green, Blue) aka (R,G,B) values. Assume there are N nodes in total.

   You can choose how to place the N nodes on the plane relative to the origin. For example, a node location can be given by $(cos\theta, sin\theta)$ where θ is the length of the curve or angle relative to the positive x-axis in radians.



In that case, all the N-1 nodes are placed on the right half of a unit circle. That is,

$$\boldsymbol{x^2 + y^2 = 1\left(-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}\right)}$$

The angle between adjacent nodes is then given by $\frac{\pi}{N-2}$. Hence, the nodes are located at ***(x, y) = (cos θ, sin θ)***, where $\boldsymbol{\theta} = \left(\frac{\pi}{2} - (\mathbf{i} - \mathbf{1}) * \frac{\pi}{\mathrm{N}-2}\right)$ and ***i=1, 2, … N-1.***

Each node is assigned a random color based on random (R, G, B) values, where $R, G, B \in [0, 255]$. You will generate node colors that are closer to one of the red, green, or blue colors. To achieve this, you will generate colors using the following rule:

> The major color value will be a random value in the range [200, 255] while the two minor color values will be a random value in the range [0, 80]. For instance, if you are generating a node that is closer to red node, the node’s R ∈ [200, 255], G ∈ [0, 80], and B ∈ [0, 80].

Following the above rule, you will also ensure that you generate equal number of nodes belonging to each color i.e. you will generate N/3 nodes that are closer to Red, N/3 nodes closer to green and N/3 nodes closer to blue. This will make sure that your network has sufficient nodes to have k neighbors of closer colors at the end of the algorithm. You must arrange these nodes randomly in the ring i.e. the nodes of similar color must not be grouped next to each other in the ring.

Now, the distance between two nodes s and t can be computed using the following equations. Even though it might seem that the (R, G, B) values can be used directly to compute the distance between two nodes, it must be noted that RGB is not perceptually uniform i.e. the distance computed using Euclidean distance on RGB might not match with the human perceived distance between colors. Thus, in this assignment you will use the below transformations to convert the color from RGB color space to L*ab color space to compute the distance between two colors.

<u>Steps to compute color distance between two nodes **s** and **t**:</u>

Note: You must perform these two steps for each node towards computing the color-distance between two nodes

Step 1: Transform RGB to XYZ

sR, sG and sB variables refer to the node’s R, G and B value.

```
var_R = ( sR / 255 )
var_G = ( sG / 255 )
var_B = ( sB / 255 )
if ( var_R > 0.04045 ) var_R = ( ( var_R + 0.055 ) / 1.055 ) ^ 2.4
else                   var_R = var_R / 12.92
if ( var_G > 0.04045 ) var_G = ( ( var_G + 0.055 ) / 1.055 ) ^ 2.4
else                   var_G = var_G / 12.92
if ( var_B > 0.04045 ) var_B = ( ( var_B + 0.055 ) / 1.055 ) ^ 2.4
else                   var_B = var_B / 12.92
var_R = var_R * 100
var_G = var_G * 100
var_B = var_B * 100
```

```
X = var_R * 0.4124 + var_G * 0.3576 + var_B * 0.1805
Y = var_R * 0.2126 + var_G * 0.7152 + var_B * 0.0722
Z = var_R * 0.0193 + var_G * 0.1192 + var_B * 0.9505
```

Step 2: Transform XYZ to CIE-L*ab:

In this step, you will use the X, Y, and Z values obtained from step 1.

Use the following values for Reference-X, Reference-Y, and Reference-Z variables:

```
Reference-X = 94.811
Reference-Y = 100.00
Reference-Z = 107.304
```

```
var_X = X / Reference-X
var_Y = Y / Reference-Y
var_Z = Z / Reference-Z
if ( var_X > 0.008856 ) var_X = var_X ^ ( 1/3 )
else                    var_X = ( 7.787 * var_X ) + ( 16 / 116 )
if ( var_Y > 0.008856 ) var_Y = var_Y ^ ( 1/3 )
else                    var_Y = ( 7.787 * var_Y ) + ( 16 / 116 )
if ( var_Z > 0.008856 ) var_Z = var_Z ^ ( 1/3 )
else                    var_Z = ( 7.787 * var_Z ) + ( 16 / 116 )

CIE-L* = ( 116 * var_Y ) - 16
CIE-a* = 500 * ( var_X - var_Y )
CIE-b* = 200 * ( var_Y - var_Z )
```

Step 3: Computing distance between s and t:

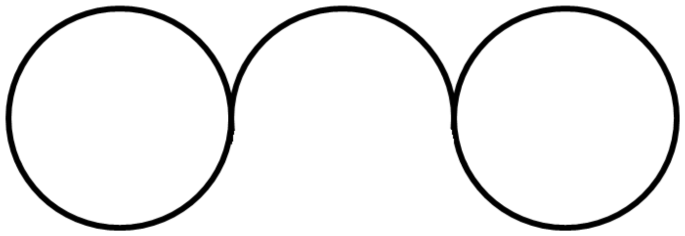
In this step you will use the CIE-L*, CIE-a*, CIE-b* obtained from step 2 towards the computation of the distance. In the following equation, L_s, a_s, b_s refers to the CIE-L*, CIE-a*, and CIE-b* values of node *s* while L_t, a_t, b_t refers to the CIE-L*, CIE-a*, and CIE-b* values of node *t*.

$$\Delta = \sqrt{(L_t - L_s)^2 + (a_t - a_s)^2 + (b_t - b_s)^2}$$

The Δ gives the color-distance between the two nodes s and t that can be used for comparison to identify the nearest nodes.

Running the algorithm using the above definition of distance will result in nodes being connected in a ring.

2. **(Spectacles Topology):** In this task, the nodes are homogenous i.e. of same color (black). Change your code and the definition of distance so that a network shaped as a "spectacles", as shown in the figure below, results from running the code instead of a "ring" and show the corresponding node graphs.



Explain your definition of distance for this case.
Hint: Spectacles topology can be obtained by constructing two circles connected by another semi-circle. The semi-circle touches the two circles externally.

<u>Additional Instructions:</u>

- Using your implementation of the algorithm, you must report the sum of distances of neighboring nodes during the initialization phase and after each running cycle. The sum of distances between neighboring nodes is defined as

$$\sum_{i=1}^{N} \sum_{node_j \in neighbors(node_i)}^{N} \boldsymbol{distance(node_i,\ node_j)}$$

  where neighbors ***(node$_i$)*** indicates all the nodes stored in the neighbor list of ***(node$_j$)***.

- In the homework report, you should show the results of testing your program using N=1000 nodes and k = 30 neighbors and running it for 40 cycles.

  - You can choose the radius for all the circles and the semi-circles in the topologies to report in the homework.

- You are also required to draw a two-dimensional plot showing the sum of distances between neighboring nodes after each running cycle of the initialization phase. In the plot, the vertical axis represents the sum of distances and the horizontal axis represents the number of cycles. You can use any tool of your choice to generate the plot.

<u>Questions:</u>

a. Explore the following two scenarios for the Ring topology and describe your findings: (15 points)
   i. When a node shares its neighbor-list with a neighbor, only the receiving neighbor will update its neighbors-list in each cycle.
   ii. When a node shares its neighbor-list with a neighbor, the receiving neighbor will share its neighbors-list to the sending node. Both the sender node and receiver node (neighbor) will update their neighbors-list in each cycle.

   <u>Note:</u> You must provide the code implementation for both the techniques mentioned in this question for the "Ring" topology. For TA testing purposes, you may choose any one technique to run i.e. there will be no extra parameter used by TA to choose the technique -- it is up to you to choose the technique. For the spectacles topology, you may choose any one technique and implement it, i.e. you are not required to implement both the techniques for the spectacles topology.

b. For the spectacles topology, run your program with different ***k*** values. How does the choice of ***k*** (the number of neighbors) affect your result? What is the minimum value of ***k*** to generate the spectacles topology to ensure connectedness among its three components (2 circles and semi-circle) after 40 cycles? Include the picture in the report. (10 points)

c. Can a node's neighbor list show the same node in multiple entries? (5 points)

d. Code execution and testing (50 points)
   i. Code will be tested in a machine running Ubuntu 22.04 running JRE 11 (for Java) and python 3.9 (for python).
   ii. Code will be tested against 3 test cases designed by the TA and points will be awarded for correctness of execution and the output generated.
   iii. TA will not modify the code or use a different command to execute the code. The TA will only execute *make* to install any required packages or libraries and to compile the code (if necessary).

e. Submission correctness. (20 points) Follow the instructions below carefully.

Submission (2/13/2024@5 pm): **No deadline extension requests will be considered**

1. Your source code containing the main method must be named TMAN.<extension of your program>. The names of the other supporting classes, if any, are up to your choice.

2. Comment every module/class/function in your code and use descriptive variable names.

3. Your program MUST run using the command line arguments as shown below. It must accept the input arguments ***N, k*** and *topology*, where the topology can be one of **R or S** which represents "ring-topology, and "Spectacles topology" respectively. The total number of cycles is fixed at 40 for this homework.
   Below is an example for the different languages.

   - ***Java: java TMAN N k topology***
   - ***Python: python TMAN.py N k topology***

4. The output of the program is the sum of distance for each cycle. For 1, 5, 10 and 15 cycles the program produces the node graph files (png, jpg, gif, …) and a file that contains the neighbor list for each node. Any output file name should have a prefix - <topology>_N<number of nodes>_k<number of neighbors>.
   E.g., R_N100_k3

   Then, the sum of distances file is named <prefix>.txt
   E.g., R_N100_k3.txt

   The node graph file is name <prefix>_<current cycle>.<file extension>
   E.g., R_N100_k3_10.png

   The neighbor list file is named <prefix>_<current cycle>.<file extension>
   E.g., R_N100_k3_10.txt

5. There are two files that you will be uploading as part of the homework submission. A tar file and a PDF file. **<u>Do not tar the PDF file.</u>**
    a. Upload the following files within a SINGLE tar file named “yourLastName_hw2.tar”:
        - The source code of your program
        - A file named “makefile” to compile the program. The TA will type only “make” to compile the program and to install any required packages or libraries. Other methods are not allowed.
        - A text file named “readme.txt” that precisely specifies the running environment including the operating system, language written, compiler version and any software needed to run your program. It should also describe the program structure such as files, classes and significant methods. If your source code is in multiple files, describe briefly the content of each file.
        - All these files should be tarred into a single file, i.e., the following command should work to create the tar file:

          *$> tar cvf <yourLastName>_hw2.tar **

    b. A PDF document (<yourLastName>_hw2.pdf) describing the definition of distance for smiley-ball topology and all results. Include the plots and the answers to the questions. Also, include your source code in the PDF. *The source code MUST be pasted towards the end of the PDF*. Include all your classes/modules/functions in the PDF file.

       NOTE: the PDF file **MUST NOT** be tarred. You will have two files to submit. A tar that contains your source code, readme file and a make file, and a PDF file hat contains plots, answers to questions, definition of distance for the smiley-ball and pasted text of your source code.

6. The tar file and the PDF must be uploaded to Canvas.

<u>Submission policy:</u>

- Do NOT include binary files. Use the file names as specified above. Incorrect submission formats will lead to a grade reduction (20 points).
- All submissions are expected by the deadline specified in the homework assignment. Grade is automatically reduced by **25% for every late day**.
- Make sure to test your submitted code using the tar file. If untar, make, compile or any other command needed to execute your program does NOT work, your homework grade will be **zero**.