CSCI 2202 COMPUTER MODELLING FOR SCIENTISTS

PROJECT: CALCULATING PAGERANK FOR A GRAPH DUE: 11 APR 2021, 11PM

This project guides you through the calculation of *PageRank*, the algorithm used by *Google* to rank search queries. The two graph files provided alongside this document (`tiny.txt, medium.txt`) represent two graphs of web-pages.

You will implement a model of search on the graph, the random crawler model, to compute the PageRank . You will do this in two ways: *(i)* By simulating the behaviour of a random crawler on the graph and *(ii)* By implementing a Markov Chain computation.

Each "page" on the web has links that can be followed to another page.
• 90% of the time, the crawler moves to another "page" by picking a **link** uniformly at random (with equal probability) from the current page and follows it.
•10% of the time, the crawler picks a random page from the entire web and jumps to it. (Google originally used the values: 85% and 15% respectively, when PageRank was first introduced).
To complete the PageRank calculations, follow the steps laid out below:

(1) You will need to read in one of the graph files provided `(tiny.txt, medium.txt)` as an *adjacency matrix*.
For this, create a function `adjMatrixFromFile(` *fileName* `)`. The function returns an *adjacency matrix* of the graph. The input graph files are given as a series of numbers, formatted in the following way:

(a) The first line of the file contains a single integer, `N`. This value represents the number of nodes (i.e. web-pages) this graph contains. Store this value. It will be important when creating a properly sized matrix.

(b) The file contains `N` more lines, one for each node in the graph. The lines have sets of integer pairs. Each integer pair `(a, b)` represents an edge in the graph, where there is an edge going from node `a` to node `b`. The node values will be in the range `[0, N-1]`. It is possible for the exact same pair to appear more than once.

An *adjacency matrix* is a square matrix containing values that show the distribution of edges within a graph. The adjacency matrix has dimensions $N \times N$. Each (row, column) entry: $a_{(i,j)}$ of the matrix contains the *number of edges* going $i \rightarrow j$. For example, location (1, 2) in the adjacency matrix generated from `tiny.txt` is 2, as the pair `1 2` occurs twice in the graph.

The `adjMatrixFromFile(fileName)` (the matrix read-in) function opens and reads a file `fileName`, creates an empty `N x N` matrix, then uses the contents of `fileName` to populate the adjacency matrix with data. You may find this easiest to do by reading the file line-by-line and using the `split()` function to break the line into a list of integers. Once the adjacency matrix is populated, close the file and return the adjacency matrix.

(2) Next, create a function `outDegrees(adjacencyMatrix)` to generate an array of the *out-degree* of each node. The out-degree of a node is the number of edges `which leave that node` (*i.e.* the number of out-links in a web-page). This function accepts a single parameter, `adjacencyMatrix`, which is an $N \times N$ NumPy matrix. The out-degree of a node `i` is equal to the sum of its row in an adjacency matrix. Create a `N x 1` array (recall the dimensions of a matrix: $row \times column$) which stores the out-degree of each node in the input graph. Return the array of out-degrees.

(3) Create a function `transitionProbabilities( adjacencyMatrix, outDegrees)` to generate and return the `transitionMatrix`. This in an $N \times N$ matrix with entries $p_{i,j}$ - which is the probability of the crawler moving: $i \to j$ defined as:

$$p_{i,j} = 0.90 \times A_{i,j}/O_i + 0.10/N$$

where A is the $(N \times N)$ adjacency matrix and O the (N) array of out-degrees. Compute and return the p matrix. Now you can proceed with computing the PageRank of each page in the graph:

I. PageRank by Simulation:

Using the functions you have implemented above, write a program `randomCrawler.py`. The program should take an integer `numTrials`, input by the user. The simulation starts the crawler at web-page 0. The crawler makes random transitions to other pages, keeping count of the number of times it visits each page.

Suppose the crawler is on page k. The transition probabilities from page k to every other page in the graph is given by the elements of the k^{th} row of the p (transition matrix).

To compute the page the crawler lands on:

- Pick a random number: $0 \leq r < 1$.
- Sum the probabilities till $\sum p[k][j] >= r$.
- The crawler makes a transition to page j.
- Increment the count of visits to page j.

Repeat the above steps `numTrials` times. The elements of the count of visits array divided by `numTrials`, gives an approximation to the PageRank for each page in the graph.

II. PageRank using the Power Method:

For this part, do the familiar Markov Chain calculation:

- Start with an initial vector $x_0 = [1, 0, 0 \ldots 0]^T$ (the superscript ^{T} indicates transpose). x_0 is a column vector ($N \times 1$ array).
- Carry out the computations: $x_{k+1} = p^T x_k$ for $k = 0, 1 \ldots$ `numTrials`.
- The final array $x_{numTrials}$ is the PageRank.

Problems:

(1) Test your programs using `tiny.txt` with `numTrials = 100`. The output is provided alongside this document.

(2) Repeat for `numTrials = 100000`. Make properly labelled histograms of the PageRanks for `tiny.txt` & `medium.txt`, for each of the above two methods.

(3) Modify the graphs to ignore the effects of multiple links. That is, if there are multiple links from a page to another, count them as a single link. Do this with `tiny.txt` and `medium.txt`. Use the new transition matrix, compute the modified PageRank vector using the Power Method only. Include the histograms and comment on the changes you notice.

(4) Finally, modify the file: `medium.txt` to add links *to* page 23 from every other page. Give the modified file a different name and save it. Run the Markov Chain calculation and make a properly labelled histogram of the PageRanks for the graph.

Note: Ideally, you will use a Jupyter notebook to present this project. You have used Jupyter notebooks in earlier labs. A reference for getting started with Jupyter notebooks is found at:
https://www.dataquest.io/blog/jupyter-notebook-tutorial/