**Assignment 1**

Graphs Algorithms and Mining

GAM Group 12

|  |  |
| --- | --- |
| **Name** | **BITS ID** |
| D M Pravin Kumar | 2022DA04333 |
| Parveen Kumar |  |

**Question 1**

**Approach:**

In the ERModel the inputs is the n, p

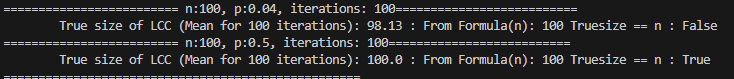
An adjacency matrix is generated by randomly choosing between values 0,1 based on the probability input provided by the user. The 1 is chosen with probability p and 0 with probability 1-p. First a upper triangular matrix is constructed and then a symmetric matrix is made by adding the transpose of upper triangular matrix to itself to form the final undirected adjacency matrix.

The edge list is generated from adjacency matrix by getting the i,j indexs of all the ones in the adjacency matrix and making a list of tuples and the adjacency list is generated by taking the adjacency matrix as input and creating a dictionary in each row of the matrix where the value is equal to 1

The largest component is generated by using the networkx module, the function nx.connected\_Components. Subgraph will give all the subgraphs and the max function will call out the max subgraph amongst them

The get\_triangles returns the number of triangles by using the inbuilt function nx.triangles(self.g).values().

**Result**

****

=========== n:100, p:0.04, iterations: 100===============

True size of LCC (Mean for 100 iterations): 98.13 : From Formula(n): 100 Truesize == n : False

===============n:100, p:0.5, iterations: 100================

True size of LCC (Mean for 100 iterations): 100.0 : From Formula(n): 100 Truesize == n : True

**Justification**

As per the theorm by Erdos-Reyni, since the np value in both cases is 4, 50 and are greater than 1 => the largest subgraph approaches the value of 1, on running 100 iterations and taking the average of True size of Largest Connected Component (LCC) is close to n (100)

**Question 2**

**Approach**

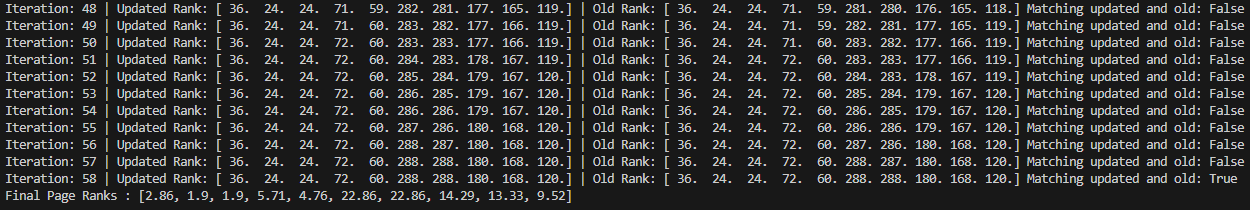
The PageRank class takes the inputs the adjacency matrix, iterations, d and n\_surfers. Initially the page rank is created by creating an array representing the page ranks(called surf) and then multiplying the number of n\_surfers to it to produce the initial list. This will be updated in each iteration.

The edge list and adjacency list are derived from the input adjacency matrix similar to how they were created in the ERModel Class. A parent list is created similar to an adjacency list but instead noting the parents of the element.

The iterate pagerank is the main function that would iterate until the number of iterations are met or there is convergence (old ranks matching the new ranks). This is achieved by using a nested for loop. The outer loop loops through the number of iterations and in the first iteration the weights are set to the number of surfers for each node. (100 in this example).

In the inner loop for every node in the node list we get the list of children and try to divide the available surfers with all the children equally, if there is a float it is moved to the next integer value. For each of the children for the node the new rank is updated. This is repeated for each node and this will give the new rank score. This is compared with the old score, if it matches the iteration is broken. If it does not match the iteration is repeated until the number of iterations are met.

**Result**



Iteration: 55 | Updated Rank: [ 36. 24. 24. 72. 60. 287. 286. 180. 168. 120.] | Old Rank: [ 36. 24. 24. 72. 60. 286. 286. 179. 167. 120.] Matching updated and old: False

Iteration: 56 | Updated Rank: [ 36. 24. 24. 72. 60. 288. 287. 180. 168. 120.] | Old Rank: [ 36. 24. 24. 72. 60. 287. 286. 180. 168. 120.] Matching updated and old: False

Iteration: 57 | Updated Rank: [ 36. 24. 24. 72. 60. 288. 288. 180. 168. 120.] | Old Rank: [ 36. 24. 24. 72. 60. 288. 287. 180. 168. 120.] Matching updated and old: False

Iteration: 58 | Updated Rank: [ 36. 24. 24. 72. 60. 288. 288. 180. 168. 120.] | Old Rank: [ 36. 24. 24. 72. 60. 288. 288. 180. 168. 120.] Matching updated and old: True

Final Page Ranks: **[2.86, 1.9, 1.9, 5.71, 4.76, 22.86, 22.86, 14.29, 13.33, 9.52]**

**The final page ranks is**

|  |  |  |  |
| --- | --- | --- | --- |
| **Node** | **Page Rank** | **Node** | **Page Rank** |
| A | 2.86% | F | 22.86% |
| B | 1.90% | G | 22.86% |
| C | 1.90% | H | 14.29% |
| D | 5.71% | I | 13.33% |
| E | 4.76% | J | 9.52% |

**Justification**

The pagerank will run either until the number of iterations complete or new ranks are similar to old ranks. The ranks start getting close from iteration 10 and matches at iteration 58. The result is that the pages F, G are the ones that have major visitors from the random walk, each accounting to about 22.86% of the population.