GRAPH MINING ASSIGNMENT

PART - 2

By: Aayush Keval Shah
2019A7PS0137H

About the assignment:

Language used: C++
The folder contains Five folders, one for each question and also the input graph files containing the edge lists for each graph.

Contents:

Problem-1: Compute the clustering coefficient of graph-1 and graph-2 from Lab-1. 2

Problem-2: A k-core is a subgraph g in the original graph G=(V, E) such that (1) g is connected, (2) induced degree of each vertex in g is at least k, and (3) g is maximal. The core number of a vertex is the maximum k such that it is part of a k-core. Compute the core number of every vertex in graph-1 and graph-2. Report the maximum core number of each graph. 3

Problem-3: Execute PageRank algorithm on the given network. Report the top 10 ranked pages. https://drive.google.com/drive/folders/1lDW7LlgEDTBTLBC6qpORxatxLDEmRfei?usp=sharing] 4

Problem-4: Compute the truss decomposition on a network and report the maximum truss number for graph-1 and graph-2. For doing this, implement Algorithm-2 in the paper attached. The source code of the algorithm is available as provided in the link. Your task is to use the CSR graph representation instead of the adjacency list representation for the implementation. You can verify your solution with the code attached 6

Problem-1: Compute the clustering coefficient of graph-1 and graph-2 from Lab-1.

sol)
The Part_2_Q1 folder contains the source code for this question. The 1_Clustering_Coef.cpp file contains the code for generating the clustering coefficient of the graph

The following functions are used:

1. **readEdges()** : Reads the edges present in input file and stores them in a 2D vector.
2. **construct_adjList(vector<vector<int>> &edges)** : Constructs the adjacency list representation from the edge list stored.
3. **display_adjList(map<int,vector<int>> adjList)** : Displays the Adjacency List.
4. **DFS(map<int,vector<int>> &adjList, vector<bool> &visited, int n, int vert, int start, int &count) :** With DFS find every possible path of length n-1 (n-1 because the last edge will be closing edge for the cycle) for a particular source. Then we check if this path ends with the vertex it started with, if yes then we count this as the cycle of length n.
5. **countCycles(map<int,vector<int>> &adjList, int n) :** Every possible path of length (n-1) can be searched using only V – (n – 1) vertices instead of all V vertices (where V is the total number of vertices) as these vertices cover the cycles of remaining vertices as well and thus maxmizies efficiency. We call DFS for each of these nodes. We consider all the paths of length n containing the node u where u belongs to V and then once done mark it as visited.
6. **int countTriplets(map<int,vector<int>> &adjList)** : Counts the number of total triplets including the cycles and the linear triplets

7. All the above functions are called in the main function chronologically and thus the Graph representations are generated.

Overall we find the cycles of given length from each node and then count the total number of cycles. The overall time taken by this program for a graph with V vertices and E edges is O(V*(V+E)) which can be written as O(V*E) as we are calling a simple DFS function for all nodes and then traversing all the edges from it. When the graph is dense the edges E will be of the order of O(V^2) and thus the total runtime of the algorithm would be O(V^3).

Thus the clustering coefficient is calculated for the given graph.

Problem-2: A k-core is a subgraph g in the original graph G=(V, E) such that (1) g is connected, (2) induced degree of each vertex in g is at least k, and (3) g is maximal. The core number of a vertex is the maximum k such that it is part of a k-core. Compute the core number of every vertex in graph-1 and graph-2. Report the maximum core number of each graph.

sol)
The Part_1_Q2 folder contains the source code for this question.

1. The 2_subgraph.cpp file contains the code for calculating the K core for the graph. The following functions are used:
    a. **readEdges()** : Reads the edges present in input file and stores them in a 2D vector.
    b. **construct_adjList(vector<vector<int>> &edges)** : Constructs the adjacency list representation from the edge list stored.

c. **display_adjList(map<int,vector<int>> adjList)** : Displays the Adjacency List.
d. **bool DFS(int v, vector<bool>& visited, vector<int> &vDegree, int k, map<int,vector<int> >& adjList)**: Helps to calculate the K core by removing the nodes with degree less than k by recursively traversing through the connected component.
e. **void printKCores(int k, int V, map<int,vector<int> >& adjList, int &maximumCore)**: Utility functions uses the DFS to traverse over all the connected components and thus calculates the maximum core.
f. All the above functions are called in the main function chronologically.

Problem-3: Execute PageRank algorithm on the given network. Report the top 10 ranked pages.

[https://drive.google.com/drive/folders/1IDW7LIgEDTBTLBC6qpORxatxLDEmRfei?usp=sharing]

sol) The Part_1_Q3 folder contains the source code for this question.

1. The 3_Pagerank.cpp file contains the code for calculating the top 10 facebook pages from the given data using the Pagerank algorithm and an arbitrary dampening factor. The following functions are used:
    a. **readEdges()** : Reads the edges present in input file and stores them in a 2D vector.
    b. **construct_adjList(vector<vector<int>> &edges)** : Constructs the adjacency list representation from the edge list stored.
    c. **display_adjList(map<int,vector<int>> adjList)** : Displays the Adjacency List.

d. **void pagerank_pass(map<int,vector<int>> &adjList, vector<pair<int,float>> &PRT, vector<int> &CT, float dampening_factor)** : It iterates over each node which has back links (in bounding links) and calculates the new pagerank for that page using the other nodes vote distributed among their out bounding links.
e. **vector<vector<string>> readData()** : it reads the data from the target input file to get the page details from the node id.
f. **bool sortbysec(pair<int,float> &a, pair<int,float> &b)**: to sort a vector of pair according to the second value in the pair
g. All the above functions are called in the main function chronologically. We run many pagerank passes till the values of pageranks start normalizing.

NOTE: the output i.e, the top 10 pages displayed are greatly dependant on the dampening factor used throughout the pagerank calculation. One can see that there can be significant changes in the pageranks if we alter the dampening factor.

As Pagerank is an iterative algorithm as far as we choose appropriate dampening factor we will see that after some finite interactions the pageranks will start to converge to a fixed number. Thus essentially we are iterating over the adjacency list created to point all the parents to a given node. Thus the Time complexity would be O(V+E) or for a dense graph it will be O(V^2) where G is a graph such that G = G(V,E).

Problem-4: Compute the truss decomposition on a network and report the maximum truss number for graph-1 and graph-2. For doing this, implement Algorithm-2 in the paper attached. The source code of the algorithm is available as provided in the link. Your task is to use the CSR graph representation instead of the adjacency list representation for the implementation. You can verify your solution with the code attached

sol)

The Part_1_Q4 folder contains the source code for this question.

2. The 4_truss_CSR.cpp file contains the code for calculating the truss decomposition of the given graph. It contains many functions and other data to generate a graph-out.txt which contains the number of triangles and other data.