COMP2521: Assignment 2
Social Network Analysis

The specification may change. A notice on the class web page will be posted after each revision, so please check class notice board frequently.

Change log:

- (12:20am Fri 25 May):
    - Closeness Centrality: description of Wasesserman and Faust formula is now moved from the FAQ to the specs, with few clarifications, consistent with the worked example already available in the FAQ.
    - Betweenness Centrality: A simpler approach to calculate betweenness centrality is introduced (same as used in the sample tests). You need to implement one of the two approaches (new or old).
    - **Assessment Criteria** is added, to help you decide where to focus your efforts.
    - The header of the function `LanceWilliamsHAC` should return `Dendrogram` (and not *Dendrogram, there was a typo). The function `freeDendrogram` is added. Download the updated file : LanceWilliamsHAC.h.
    - Download the latest Ass2_Testing.zip.
- 10:20am Sun 20 May:
    - "Part-4: Discovering Community" is now available. As discussed in the Friday lecture, you may want to first watch three videos, make sure you watch them in the sequence, don't jump to the last video.
    - You should also read Zain's latest COMP2521 Ass2 FAQ (18s1). If you have any questions or need clarifications, please send him an email at z.afzal@unsw.edu.au , clearly write "comp2521 Ass2" in the subject line.
- 07:30am Tue 15 May:
    - Zain maintains the FAQ for this assignment at the following link: COMP2521 Ass2 FAQ (18s1)
      If you have any questions or need clarifications, please send him an email at z.afzal@unsw.edu.au , clearly write "comp2521 Ass2" in the subject line.
    - Change in the way you need to calculate "Closeness", see Part-3.
    - Added section on "Testing"

Objectives

- to implement graph based data analysis functions (ADTs) to mine a given social network.
- to give you further practice with C and data structures (Graph ADT)
- to give you experience working in a team

Admin

Marks	20 marks (scaled to 14 marks towards total course mark)
Group	This assignment is completed in **group of two**, based on your current lab group.
Due	5pm Friday Week-13
Late Penalty	2 marks per day off the ceiling. Last day to submit this assignment is 5pm Tuesday Week-14, of course with late penalty.
Submit	Read instructions in the "Submission" section below.

Aim

In this assignment, your task is to implement graph based data analysis functions (ADTs) to mine a given social network. For example, detect say "influenciers", "followers", "communities", etc. in a given social network. You should start by reading the wikipedia entries on these topics. Later I will also discuss these topics in the lecture.

- Social network analysis
- Centrality

The main focus of this assignment is to read graph data from a given file, build a directed weighted graph, calculate measures that could identify say "influenciers", "followers", etc., and also discover possible "communities" in a given social network.

Code from the course material

For this assignmet you can use soruce code that is available as a part of the course material (lectures, exercises, tutes and labs). However, you must properly acknowledge it in your solution.

Input Data

You need to read graph data from a text file, and build a directed weighted graph using the graph ADT you will be implementing for this assignment, see below. In an input file, each line represents one directed weighted edge. On each line three values representing source, destination and weight are separated by a comma. Note that there may be additional spaces before or after a comma.

```
0, 4, 5
1,   2  , 14
  2  , 4, 3
3, 2, 10
2,1,6
```

The above input file represents a graph with five directed weighted edges. The first line represents the edge from node '0' to node '4' with weight '5', the second line represents the edge from '1' to '2' with weight '14', and so on.

Part-1: Graph ADT

You need to first implement a graph ADT, using Adjacency List Representation as discussed in the lectures. Please note that the graph ADT provided uses Adjacency Matrix Representation, and you need to use Adjacency List Representation for this assignment as discussed in the lectures. You must **not** use any other representation, even if they are efficient or not! please don't copy from many possible implementations available on the web ;-) Remenber, you are here to learn ;-)

Your task: In this section, you need to implement the following file:

- `Graph.c` that implements all the functions defined in `Graph.h`.

Part-2: Dijkstra's algorithm

In order to discover say "influencers", we need to **repeatedly** find shortest paths between **all pairs** of nodes. In this section, you need to implement Dijkstra's algorithm to discover shortest paths from a given source to all other nodes in the graph. Considering we will be using this operation (function) repeatedly, the time complexity of your function for Dijkstra's algorithm should be O(E + V log V), as discussed in the lectures. To achieve this goal, you need to implement a priority queue ADT with time complexity of O(log n). Initially you could implement a simple priority queue with time complexity of O(n), however, later before your submission you should try to implement your priority queue ADT with the time complexity of O(log n) to receive full marks for this part.

Your task: In this section, you need to implement the following two files:

- `PQ.c` that implements all the functions defined in `PQ.h`.
- `Dijkstra.c` that implements all the functions defined in `Dijkstra.h`.

Part-3: Centrality Measures for Social Network Analysis

Centrality measures play very important role in analysing a social network. For example, nodes with higher "betweenness" measure often correspond to "influencers" in the given social network. In this part you will implement few well known centrality measures for a given directed weighted graph.

Descriptions of some of the following items are from Wikipedia at Centrality, adapted for this assignment.

Degree Centrality

Degree centrality is defined as the number of links incident upon a node (i.e., the number of ties that a node has). The degree can be interpreted in terms of the immediate risk of a node for catching whatever is flowing through the network (such as a virus, or some information). In the case of a directed network (where ties have direction), we usually define two separate measures of degree centrality, namely indegree and outdegree. Accordingly, indegree is a count of the number of ties directed to the node and outdegree is the number of ties that the node directs to others. When ties are associated to some positive aspects such as friendship or collaboration, indegree is often interpreted as a form of popularity, and outdegree as gregariousness.

For a given directed graph $G := (V, E)$ with $|V|$ vertices and $|E|$ edges,

- the indegree centrality of a vertex v is defined as $C_{Din}(v) = inDegree(v)$.
- the outdegree centrality of a vertex v is defined as $C_{Dout}(v) = outDegree(v)$.

Closeness Centrality

Closeness centrality (or closeness) of a node is calculated as the sum of the length of the shortest paths between the node (x) and all other nodes ($y \in V \wedge y \neq x$) in the graph. Generally closeness is defined as below,

$$C(x) = \frac{1}{\sum_y d(y, x)}.$$

where $d(y, x)$ is the shortest distance between vertices x and y.

However, considering most likely we will have isolated nodes, for this assignment you need to **use Wasserman and Faust formula** to calculate closeness of a node in a directed graph as described below:

$$C_{WF}(u) = \frac{n-1}{N-1} * \frac{n-1}{\sum_{v=0}^{n-1} d(u, v)}.$$

where $d(u, v)$ is the shortest-path distance in a directed graph from vertex u to v, n is the number of nodes that u can reach, and N denote the number of nodes in the graph. Please see the example in the FAQ, at What on earth is this math, i'm not a math person

Based on the above, the more central a node is, the closer it is to all other nodes. For for information, see Wikipedia entry on Closeness centrality.

Betweenness Centrality

The betweenness centrality of a node v is given by the expression:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v.

The following are two alternative approaches to calculate normalised betweenness centrality. You need to implement one of the following two approaches, you will be awarded equal marks for either of the following two. We recommend the first approach, it is easier! and also avoids zero as denominator (for n>2). The second approach is from the Wikipedia entry, which was listed earlier.

Approach-1 (preferred, used by the sample tests):

$$\text{normal}(g(v)) = \frac{1}{((n-1)(n-2))} * g(v)$$

where, n represents the number of nodes in the graph.

Approach-2 (from Wikipedia):

$$\text{normal}(g(v)) = \frac{g(v) - \min(g)}{\max(g) - \min(g)}$$

which results in $\max(normal) = 1$ and $\min(normal) = 0$

Your task: In this section, you need to implement the following file:

- `CentralityMeasures.c` that implements all the functions defined in `CentralityMeasures.h`.

For more information, see Wikipedia entry on Betweenness centrality

Part-4: Discovering Community

In this part you need to implement the Hierarchical Agglomerative Clustering (HAC) algorithm to discover communities in a given graph. In particular, you need to implement Lance-Williams algorithm, as described below. In the lecture we will discuss how this algorithm works, and what you need to do to implement it. You may find the following document/video useful for this part:

- Hierarchical Clustering (Wikipedia), for this assignment we are interested in only "agglomerative" approach.
- Brief overview of algorithms for hierarchical clustering, including Lance-Williams approach (pdf file).
- Three videos by Victor Lavrenko, watch in sequence!
    - Agglomerative Clustering: how it works
    - Hierarchical Clustering 3: single-link vs. complete-link
    - Hierarchical Clustering 4: the **Lance-Williams algorithm**

Distance measure: For this assignment, we calculate distance between a pair of vertices as follow: Let wt represents maximum edge weight of all available weighted edges between a pair of vertices v and w. Distance d between vertices v and w is defined as $d = 1/wt$. If v and w are not connected, d is infinite.

For example, if there is one directed link between v and w with weight wt, the distance between them is $1/wt$. If there are two links, between v and w, we take maximum of the two weights and the distance between them is $1/max(wt_{vw}, wt_{wv})$. Please note that, one can also consider alternative approaches, like take average, min, etc. However, we need to pick one approach for this assignment and we will use the above distance measure.

You need to use the following (adapted) Lance-Williams HAC Algorithm to derive a dendrogram:

```
• Calculate distances between each pair of vertices as described above.
• Create clusters for every vertex $i$, say $c_i$.
• Let $Dist(c_i, c_j)$ represents the distance between cluster $c_i$ and $c_j$, initially it represents distance between vertex $i$ and $j$ .
• For k = 1 to N-1
        - Find two closest clusters, say $c_i$ and $c_j$. If there are multiple alternatives, you can select any one of the pairs of closest clusters.
        - Remove clusters $c_i$ and $c_j$ from the collection of clusters and add a new cluster $c_{ij}$ (with all vertices in $c_i$ and $c_j$) to the collection of clusters.
        - Update dendrogram.
        - Update distances, say $Dist(c_{ij}, c_k)$, between the newly added cluster $c_{ij}$ and the rest of the clusters ($c_k$) in the collection using Lance-Williams formula using the selected method ('Single linkage' or 'Complete linkage' - see below).
• End For
• Return dendrogram
```

Lance-Williams formula:

$$Dist(c_{ij}, c_k) = \alpha_i * Dist(c_i, c_k) + \alpha_j * Dist(c_j, c_k) + \beta * Dist(c_i, c_j) + \gamma * abs(Dist(c_i, c_k) - Dist(c_j, c_k))$$

where α_i, α_j, β, and γ define the agglomerative criterion.

For the *Single link method*, these values are: $\alpha_i = 1/2$, $\alpha_j = 1/2$, $\beta = 0$, and $\gamma = -1/2$. Using these values, the formula for Single link method is:

$$Dist(c_{ij}, c_k) = 1/2 * Dist(c_i, c_k) + 1/2 * Dist(c_j, c_k) - 1/2 * abs(Dist(c_i, c_k) - Dist(c_j, c_k))$$

We can simplify the above and re-write the formula for **Single link method** as below

$$Dist(c_{ij}, c_k) = min(Dist(c_i, c_k), Dist(c_j, c_k))$$

For the *Complete link method*, the values are: $\alpha_i = 1/2$, $\alpha_j = 1/2$, $\beta = 0$, and $\gamma = 1/2$. Using these values, the formula for Complete link method is:

$$Dist(c_{ij}, c_k) = 1/2 * Dist(c_i, c_k) + 1/2 * Dist(c_j, c_k) + 1/2 * abs(Dist(c_i, c_k) - Dist(c_j, c_k))$$

We can simplify the above and re-write the formula for **Complete link method** as below

$$Dist(c_{ij}, c_k) = max(Dist(c_i, c_k), Dist(c_j, c_k))$$

Your task: In this section, you need to implement the following file, download updated file:

- `LanceWilliamsHAC.c` that implements all the functions defined in `LanceWilliamsHAC.h`.

Assessment Criteria

- Part-1: Graph ADT (7%)
- Part-2: Dijkstra's algorithm
  - PQ 8% (max 6% for O(n) time complexity). You loose only 2% marks, that is 0.4/20 if you implement PQ with O(n), so only if you like the challenge, use say balanced trees to get O(log n) time complexity.
  - Dijkstra 15%
- Part-3:
  - Degree Centrality (5% marks),
  - Closeness Centrality (15% marks),
  - Betweenness Centrality (15% marks)
- Part-4: Discovering Community (15% marks)
- Style, Comments and Complexity: 20%

Testing

You can download the following zip file to test Part-2 and 3 : Ass2_Testing.zip. Please **replace the required *.c files** with your solution *.c files.

Instructions on how to use the above testing framework is available at (on the faq at) **How do i use the testing interface**.

Submission

To be added later...

Plagiarism

This is a group assignment. Each group will have to develop their own solution without help from other people. In particular, it is not permitted to exchange code or pseudocode. You are not allowed to use code developed by persons other than your group member. If you have questions about the assignment, ask your tutor. All work submitted for assessment must be entirely your own work. We regard unacknowledged copying of material, in whole or part, as an extremely serious offence. For further information, see the Course Information.

-- end --