Programming Assignment 2: Social Network Graph

CS130a Winter 2022

Due March 10, 2022

1 Backgrund

1.1 Social Network graph

When we think of a social network, we think of Facebook, Twitter. This kind of network is representative of the broader class of networks called 'social'. There are three essential characteristics of a social network.

- There is a collection of entities that participate in the networks, which are normal persons.
- There is at least one relationship between entities of the network.
- There is an assumption of locality, which means if entity A is related to both B and C, then there is a higher probability than average that B and C are related.

Social networks are naturally modeled as undirected graphs, which are sometimes referred to as social graphs. Persons are the nodes and an edge connects two nodes if the nodes are related by the relationship, where the edge weights (normalized to the left closed right open interval between 0 and 1) indicate the strength of the tie (values close to 0 mean strong friendship). For each person, there is a vector indicating the level of interest (normalized again to lie between 0 and 1) on k different hobbies (values close to 1 mean strong interest).

Example 1: Figure 1 is an example of a tiny social network. There are five persons in the social graph, which are nodes 1 through 5. The relationship, which we might think of as 'friends', is represented by the edges with a weight between $[0, 1)$.

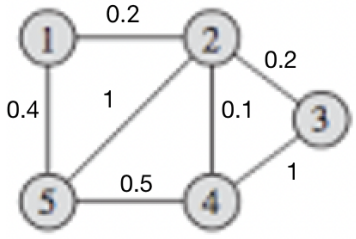

Figure 1: Example of a social network graph

1.2 Adjacency List

The premise of this project is that you are asked to search for different types of information in the social graph. As you already know there are two possible representations for maintaining graph data in memory: Adjacency Matrix and Adjacency List. The Adjacency Matrix representation although simple is not storage efficient. In general, it should be used in the context of dense graphs (i.e. for graphs that have a large number of edges). Even with the complete graph, this representation incurs 2X storage since the matrix maintains redundant information(Keep in mind that whenever we use the general term "graph", by default we mean an undirected graph). Thus the storage utilization never exceeds more than 50%. Thus for most graph applications, especially in the context of sparse graphs such as social networks, the adjacency list representation is used.

As you may recall, an adjacency list stores the vertices of a graph in an array, and for each vertex in the graph, it maintains the list of adjacent nodes as a linked list. Figure 2(a) illustrates an example of a graph and Figure 2(b) illustrates the adjacency list representation, and Figure 2(c) is the adjacency matrix representation.

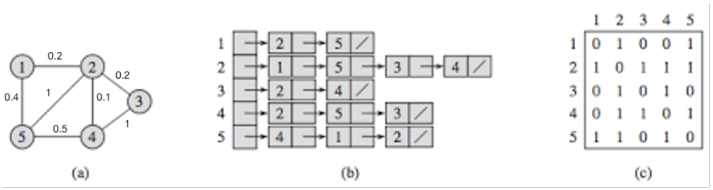

Figure 2: Example of a adjacency list

In this project, you are asked to develop an efficient representation of social graphs based on the adjacency list representation.

1.3 Connected Components

A connected component or simply component of an undirected graph is a sub-graph in which each pair of nodes is connected via a path. In other words, a set of nodes forms a connected component in an undirected graph if any node from the set of nodes can reach any other node by traversing edges.

In this assignment, you will be asked to calculate the diameter, radius and center of each connected component. Let $G = (V, E, w)$ be a graph. For a vertex v, the eccentricity of v is the maximum of the distance to any vertex in the graph, i.e., $\max_{u \in V} \{d_G(v, u)\}$. The diameter of a graph is the maximum of the eccentricity of any vertex in the graph. In other words, the diameter is the longest distance between any two vertices in the graph.

The radius of a graph is the minimum eccentricity among all vertices in the graph, and a center of a graph is a vertex with eccentricity equal to the radius. For a general graph, there may be several centers and a center is not necessarily on a diameter. For example, in Figure 3, the shortest path between v_1 and v_4 is a diameter of length 6 . Meanwhile v_2 and v_3 are endpoints of another diameter. The four vertices represented by white circles are centers of the graph, and the radius is 4. However, in this programming assignment, the problem we consider is on a weighted tree. So, when you calculate the diameter, radius, and a center for the test case, don't forget to consider the weight of each edge.

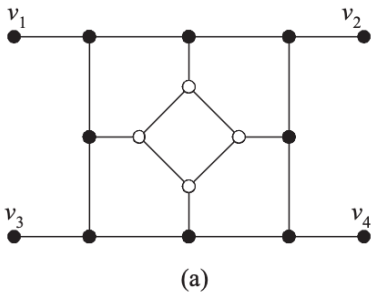

Figure 3: Diameters, centers and radius of an unweighted graph

2 Implementation

In this project, you are asked to solve 6 questions by implementing an undirected social network graph, where the edges in the graph have their weights (normalized to the open interval between 0 and 1) indicate the strength of the tie (values close to 1 mean strong friendship). Furthermore, every person (vertex) in this graph has a vector indicating the level of interest (normalized again to lie between 0 and 1) on k different hobbies (values close to 1 mean strong interest). In this graph, you need to

develop separate functions to solve the following questions. In our test case, the vertices number are set to be $[1, 100]$ and the hobbies are set to be $[1, 20]$, where the total number of hobbies k is set to be 20.

- Find the average degree and the vertex with the highest degree
- Find the number of connected components and the diameter, radius, and a center of each component. Use the weights on edges in the computations of diameter, radius, and center.
- Find the ratio between the number of open (a simple, undirected graph consisting of three vertices and two edges) and closed triangles (a simple, undirected graph consisting of three vertices and three edges).
- Given a start node $x = 39$, a hobby $h = 7$, and a threshold $t = 0.5$, find the closest node from x with an interest level of at least t on hobby h.
- Given a hobby $h = 7$, find a person with the highest interest in h.
- For two persons a and b with interests $\{h_1^a, \cdots, h_{20}^a\}$ and $\{h_1^b, \cdots, h_{20}^b\}$. Define a hobby distance between them to be $\sqrt{(h_1^a - h_1^b)^2 + \cdots + (h_{20}^a - h_{20}^b)^2}$, which is the square root of the sum of the squares of the difference between individual hobby interests. For example, the Find a pair of nodes x and y whose ratio between hobby distance and graph distance (the sum of the weights on the path from x to y) is smallest, **if there exists a path from x to y**. Note that such a pair of nodes would benefit greatly by becoming neighbors.

You have to implement two classes in C++:

- GraphGenerator
- GraphOperator

GraphGenerator:
This class takes the list of edges as input and generates the adjacency list for maintaining the graph. The input file contains the list of edges and their weight. Each row of the input file represents an edge using the corresponding vertices. For simplicity, you can assume that the vertices are represented by non-negative integers and if the graph contains N vertices they will be numbered from 0 to N-1. The three numbers separated by a comma are the first vertex, the second vertex and the weight of the edge between these two vertices.

You need to develop an AddEdge function to insert both of the vertices connected by that edge in each other's adjacency list. For example, if an edge between (1, 3) has to be added, then vertex 1 will be stored in vertex 3's vector list and 3 will be stored in vertex 1's vector list.

Furthermore, you need to maintain a vector for each vertex indicating the level of interest (normalized again to lie between 0 and 1) on 20 different hobbies (values close to 1 mean strong interest) in the code.

GraphOperator:
This class is used to develop functions to solve the 6 questions. So here are the 6 basic functions, you are also free to develop other functions you may use to solve these problems.

- FindAverageDegree: Find the average degree and the vertex with the highest degree
- FindHighestDegree: Find the vertex with the highest degree
- FindConnectedNumber: Find the number of connected components
- FindConnectedParameters: Find the diameter, radius, and centers of each component.
- FindTrianglesRatio: Find the ratio between the number of open and closed triangles
- FindClosestNode: Find the closest node from x with an interest level of at least t on hobby h.
- FindHighestInterest: Find a person with the highest interest in h.
- FindDistanceRatio: Find a pair of nodes x and y whose ratio between hobby distance and graph distance is smallest.

3 Input and Output

3.0.1 Program Input

After implementing the above classes, you need to test them. Your program will be given a file 'edges.txt' which contains the list of edges in the graph and their weights. You can assume that we don't have an isolated vertex(a vertex with no edge). Each line in the input file has three integer values separated by comma "$integer1, integer2, integer3$". These integer values are two vertex numbers (from 1 to 100) and the weight of this edge, which together, define an edge in the graph. For example the following values represent the graph in Figure 2.a:

```
1, 5, 0.4
1, 2, 0.2
2, 5, 1.0
2, 4, 0.1
2, 3, 0.2
3, 4, 1.0
4, 5, 0.5
```

Furthermore, we have another file that contains the level of interest on k different hobbies for each person, which is called 'hobbies.txt'. Each line in the input file has k integer values separated by comma "$\cdot, \cdot, \cdots, \cdot$", which is the interests parameters for each person from 1 to 100. In our test case, we set the value of k as 20.

3.0.2 Program Output

The sample output of the program is shown below:

```
The average degree:
<the value of average degree>
The vertex with the highest degree:
<the number of vertex with the highest degree>
The number of connected components:
<the number of connected components>
The diameter, radius, and center(s) of each component:
<diameter, radius, center1, center2, ...>
...
<diameter, radius, center1, center2, ...>
The ratio between the number of open and closed triangles:
<the value of the ratio>
The closest node:
<the number of vertex>
A closest with the highest interest:
<the number of vertex>
The pair of nodes x and y:
<the number of vertex x, the number of vertex y>
```

4 Deliverable

4.1 AutoGrader

You will upload all your implementation files, test files, document and makefile to Gradescope. Before printing the diameter, radius, and centers of each component, **sort them in non-decreasing order based on the value of the diameter**. This would be helpful for autograder to compare your results with sample output. The autograder will be implemented later. **Your executable file has to be named "PA2".** Its output should follow from the **Program Output** section.

4.2 Documentation

In the document, you should explore the time complexity for each problem. To do this, you can both analyze your algorithm theoretically and test your algorithm by choosing different scales of input edges to get a conclusion. You are required to provide the setup and outputs of these test cases. Plot a diagram to show the relation between them. You need to justify your experimental results with your theoretical analysis.

4.3 Demo

Another half of you will present a demo. You will be asked to run the program, demonstrate some of the functions and also justify your implementation decisions. Note that all students who have not presented a demo for PA1 need to give the presentation this time, irrespective of their teams.