Lab 3: Graph Problems

Due date: Friday, October 27, 11:59pm.

Lab Assignment

Assignment Preparation

This is a pair programming lab.

The Dataset

For this lab you will be working with the Association Wordgame dataset. The dataset provided to you comes from Kaggle[1]. The dataset consists of a single CSV file, `wordgame_20170721.csv` that is made available to you.

The dataset consists of 334,037 lines, each line containing a pair of words or phrases. The pair represents a specific association a human player made when playing a version of a *word associations* game. The player was provided the first word in the pair, and in resposne supplied the second word.

A single line of the dataset looks as follows:

```
6075,captain,ship,WP,9
```

Each line contains the following columns:

```
author,word1,word2,source,sourceID
```

Here,

- `author` is a unique Id of a person who played the game

[1]https://www.kaggle.com/anneloes/wordgame/data

- `word1` is the word/phrase that the player received
- `word2` is the word/phrase that the player supplied in response to `word1`
- `source` and `sourceId` columns specify the source from which this data is collected

For the purposes of this assignment, *you will ignore all columns from this datset* except the `word1` and `word2` columns.

Additionally, for the purposes of this assignment you can ignore any lines where either `word1` or `word2` is a phrase rather than a single word. For example, the following line

```
4676,u2 fans,people who have understanded what the meaning of life is,U2,8
```

should be ignored and not processed by your graph construction program.

Some other lines may contain two-word combinations that may make sense to preserve, e.g., names of people:

```
4762,doug henning,david copperfield,U2,8
```

You are allowed to decide whether you want to try to keep these lines and include entries for `"doug henning"` and `"david copperfield"` in your final graph or not. This is left up to you.

The Task

For this assignment you will write a variety of programs to construct the graph represented by the Association Wordgame dataset, and analyze it in a number of ways.

Part 1. Graph Construction. Write a program that takes the `wordgame_20170721.csv` file as input and constructs two graphs represented by the ingested dataset.

In both graphs, the vertices will represent words from the dataset, and the edges will represent associations. In what follows, for the sake of simplicity we refer to *"edges between words"* where a more formal way is to say *"edges between vertices representing words"*.

Graph 1 is the undirected weighted word association graph. In this graph, there is an edge between a pair of words if the words appear in some input line of the `wordgame_20170721.csv` file *in any order.* Additionally, the weight of an edge (`word1`, `word2`) is the number of times this pair appears (in any order) in the input file.

Graph 2 is the directed weighted word association graph. In this graph, there is a directed edge from `word1` to `word2` if the pair (`word1`, `word2`) appeared on some line of the input file *in that specific order.* The weight of the edge (`word1`, `word2`) is the number of times this pair of words appears *in this specific order* in the input file.

Your first program (`buildGraphs.py` or `buildGraphs.java`) shall take `wordgame_20170721.csv` as input[2]. It shall produce as output two files, `directedGraph.<ext>` and `undirectedGraph.<ext>` where `.<ext>` is an extension that matches the format of your data (it can be XML, JSON, binary format, CSV or any other format you want). Each of the output files must contain the precise description of the graph in a form of an adjacency list.

All your other programs will take as input either `directedGraph.<ext>` or `undirectedGraph.<ext>` file.

Part 2. Connectivity Analysis. Write a program that takes as input the undirected weighted word association graph and finds the total number of connected components in it.

Your program (name it `connected.py` or `Connected.java`) shall output the following information in a *human-readable self-explanatory, and nicely formatted way*:

- Total number of connected components.
- For each connected component:
  - Total number of words in the component.
  - Highest degree of a vertex in the component, number of vertices in the component with that degree.
  - List of words in the component with the highest degree. If the highest degree is small (2 or 3) and/or there are many (more than 100) words with this degree in the component, report the first 20 words with the highest degree in the component in alphabetical order. Otherwise, report all words with the highest degree (in a compact way)

Part 3. Degree Distribution Information. Write a program (`degrees.py` or `Degrees.java`) that takes as input both the directed and the undirected graphs (one at a time) and computes the following histograms[3]:

- Distribution of the vertex degrees in the undirected graph.
- Distribution of the vertex in-degrees in the directed graph.
- Distribution of the vertex out-degrees in the directed graph.

[2]you may want to add a parameter specifying where the input file is located so that we would not need to copy this file into ever person's directory.

[3]A histogram is simply a mapping from values of a specific quantity to their frequency of occurrence. You do not have to draw any graphics for it, just report the actual frequency counts.

The output of this program shall be the three histograms properly labelled and reported in a human-readable form.

Note: Recall that a `degree` of a vertex in the graph is the number of edges touching this node. In- and out-degree, respectively, is a number of incoming/outgoing edges for a vertex in a directed graph.

Note: If you want to, you can make your `buildGraphs` program produce this output. If you do choose to implement this part of the assignment as part of the `buildGraphs` program, please let us know in the README file, and ensure that your `buildGraphs` program produces the appropriate output.

Part 4. Word Association Chains. Write a program (`wordChains.py` or `WordChains.java`) which takes as input (one at a time) a list of pairs of words and produces, for each pair of words (`word1`, `word2`) the shortest (in terms of the number of edges) path from word `word1` to word `word2` via the observed associations. If there is no path in the graph between the two words, a message specifying this shall be returned instead.

The input to your program will be a file of query word pairs. A sample file may be

```
bricks, drift
white, punk
```

Your program's output shall look similar to this[4]:

```
bricks -> fire -> wood -> drift, 3 steps
white -> teeth -> loose -> daft -> punk, 4 steps
```

By default, use the undirected graph for this assignment. **For extra credit** implement a similar search in the directed graph by allowing the user to select the type of graph in an input parameter to your program.

Part 5. The "Kevin Bacon words." The "Kevin Bacon game" is a game of connecting different actors to Kevin Bacon via shared movies (Actor X has been in a movie with Actor Y who has been in a movie with Kevin Bacon).

We want to find words in our dataset that are good candidates for being the "Kevin Bacon of words", i.e., words from which **many** other words are reachable via **reasonably short** paths (in the case of the Kevin Bacon game, the claim is that every actor is at most six steps away from Kevin Bacon).

While finding all possible candidate words would require solving the all-sources shortest paths problem (which we won't cover until later in the

[4]These are true paths in the graph, although I do not know if they are actually the shortest in terms of the number of edges.

course - so this may come as an assignment in a later lab), we can limit our search heuristically to words with high degree (the idea is that if a word has a high degree, it must be well-connected with a lot of other words).

Write a program (`bacon.py` or `Bacon.java`) that takes as input the undirected graph, and produces a list of candidate words that

- Are connected to a lot of other words[5]
- Are connected to a lot of words via relatively short paths.

For each word discovered, report (a) the average path length from this word to all other words connected to it, (b) the histogram of path lengths from this word to other words.

You may choose your own criteria for which words to report, how many words to report, how to determine which words are better than others. Please make these criteria explicit in your README file.

Deliverables

This lab has only electronic deliverables. All files (the five programs, README, and any supplementary files) shall be submitted by the assignment deadline using the following `handin` command:

Use `handin` to submit:

```
$ handin dekhtyar lab03 <files>
```

Note that we expect that all your programs will compile from the command line. Make sure you test for that.

Good Luck!

[5]If there is only one connected component, than every word would be connected to all other words.