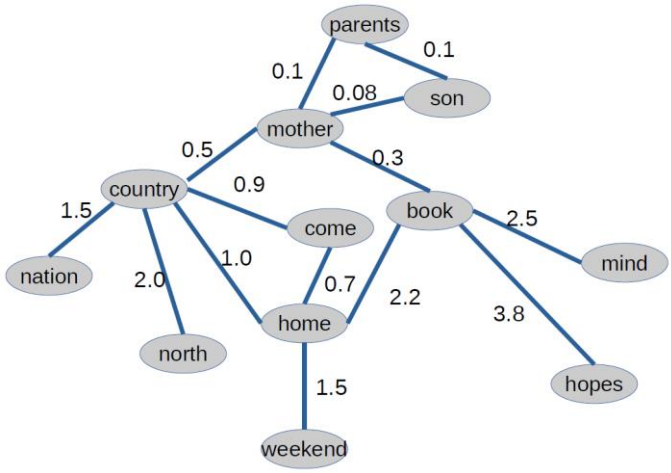
COSC 2P03 – Advanced Data Structures
Fall 2020
Assignment #4

Due Date: December 4th, 6pm No late assignments will be accepted.
This assignment accounts for 8% of your final grade and is worth a total of 80 marks.

In natural language processing, word embedding techniques have a fundamental impact to modern modelling of languages. Through applying a word embedding approach on a large text data set, words (symbolic elements) can be converted to vectors of continuous values thus allowing convenient manipulations. Formally, we can transform a list of N words $\{w_1, w_2, \ldots, w_N\}$, i.e. the vocabulary to a list of vectors of M continuous values: $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ where $\mathbf{x}_n \in R^M$, where $n = 1, 2, \ldots, N$. If you are interested in this topic, you can read this: https://en.wikipedia.org/wiki/Word_embedding. Using word vectors, we can measure the distance between any pair of words, and build an undirected word net. An example of such network with 12 words is show in the following figure, where each vertex has a word as label and the weight on each edge shows the distance of the corresponding words. (Note: weights on this figure were just assigned intuitively which is good enough to show a simple example. Thus, these weights are different from the values you will obtain from this assignment.)



In this assignment, you are asked to build a simplified *word net* where each vertex represents a word and the edge between two vertices represents the distance between the two vertices (i.e. words). Specifically, you will do the following tasks.

1. Define a class named `WordNet` to represent an undirected graph with attribute variables such as `words` (an array of words/strings), `vectors` (an N by M matrix/array), `A` (the binary adjacency matrix), `W` (the weighted adjacency matrix), `N` (number of words), and other auxiliary variables. (5 marks)

2. Within the class, define a method named `read_data` which reads the text file (`wordvector.txt`) given with this assignment. In the text file, there are 20,000 rows. For a row, the first element is a word, and the rest 50 values are the corresponding word vector. You are only required to read the 101st – 3100th rows (1-based indexing). Thus, you will have 3,000 words and corresponding vectors. In this function, you will calculate the Euclidean distance between any two words among the 3,000 words using $W_{i,j} = \sqrt[2]{\sum_{m=1}^{M}(x_{i,m} - x_{j,m})^2}$. If the distance is greater than a threshold (use 3.0 in your assignment), reset it to infinity, otherwise keep it in matrix `W` and set the corresponding value in `A` properly. You should print the number of edges in this graph on screen and show it in your PDF file. (20 marks)

3. Within the class, define a method named `find_modules` to identify the number of connected modules (subgraphs/islands) in the graph. You should print on screen the number of modules you can find and the corresponding sizes (i.e. the number of vertices in a module) (note: only print sizes of top 20 modules), and copy this output to your PDF file. Hint: you can use either the `BFS` (breath-first search) method or the `DFS` (depth-first search) method to obtain all vertices in a module. (15 marks)

4. Within the class, design a method named `find_shortest_path` to find the shortest path from a given word (say `fromword`) to another given word (say `toword`) on the largest module. For example, you are required to find the shortest paths from "money" to "future", from "village" to "city", from "bad" to "good", from "problem" to "opportunity", respectively. You are required to implement and apply the unweighted BFS method and Dijkstra's algorithms, respectively. Print the paths (in a sequence of words) on screen, and reported them in the PDF file. You should also report whether the results from the two algorithms are different. (20 marks)

5. Within the class, implement Kruskal's algorithm to find the minimum spanning tree (MST) on the largest module. In the algorithm, you should save all the edges and associated weights in the MST to a txt file (attach this txt file within your zipped folder). Your function should print the total length (i.e. sum of weighted distances) of the MST on the screen and report them in your PDF file. (20 marks)

Submission Requirements:
All submissions will be via Sakai. All of the following must be put in a single, zipped folder for submission:
1. All program (source code) files required to run your program. All code must be commented and properly documented.
2. The txt file mentioned in Task 5.
3. A "printout" of your program, which has been printed into PDF format. Ensure that the first page of this PDF document contains a statement specifying whether you used IntelliJ, Eclipse or DrJava.
4. A printout (in PDF) of the specified outputs for your program when using the supplied input file `wordvector.txt`.