Assignment #4: Poetic Walks

ECE 422C (Prof. Edison Thomaz) - The University of Texas at Austin – Spring 2022

Problem Statement:

Graphs — what are they good for? Poetry!

In this lab, we will explore a common Natural Language Processing technique known as N-gram.

Write a program that completes the poetry with bridge words.

For example, here's a small but inspirational corpus:

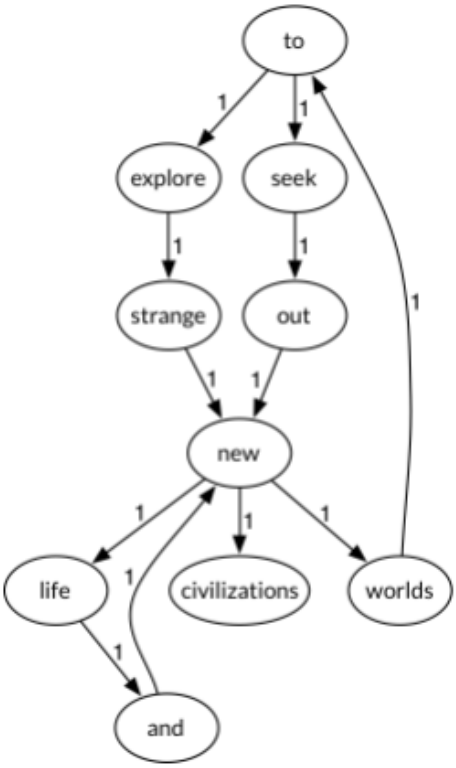
Poem Input:

To explore strange new worlds
To seek out new life and new civilizations

`GraphPoet` will use this corpus to generate a word affinity graph where:

- vertices are case-insensitive words, and
- edge weights are in-order adjacency counts.

The graph is shown below. In this example, all the edges have weight 1 because no word pair is repeated.



From there, given this dry bit of business-speak input:

Input:

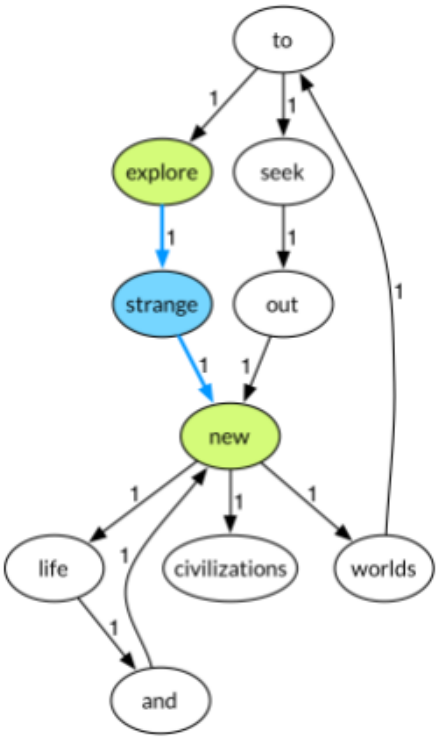
→ *Seek to explore new and exciting synergies*

The poet will use the graph to generate a poem by inserting (where possible) a *bridge word* between each pair of input words given the poem input:

The resulting output:

> Seek to explore **strange** new **life** and exciting synergies

w_1		w_2	path?	bridge word
seek	→	*to*	no two-edge-long path from *seek* to *to*	—
to	→	*explore*	no two-edge-long path from *to* to *explore*	—
explore	→	*new*	one two-edge-long path from *explore* to *new* with weight 2	*strange*
new	→	*and*	one two-edge-long path from *new* to *and* with weight 2	*life*
and	→	*exciting*	no vertex *exciting*	—
exciting	→	*synergies*	no vertices *exciting* or *synergies*	—



Input and Output Requirement:

Your function will be called in Main.java in the following format:

```
public static void main(String[] args) throws IOException {
    final GraphPoet nimoy = new GraphPoet(new File("corpus.txt"));
    System.out.println(nimoy.poem(new File("input.txt")));
}
```

Input 1:

The **constructor** of GraphPoet will take in a file called "corpus.txt" with all the bridge words. Create a Weighted Graph of these words. You can assume the corpus will be free of non-alphabetic symbols, besides spaces and newlines.

Example of "corpus.txt"

> *To explore strange new worlds*
> *To seek out new life and new civilizations*

Input 2:

In GraphPoet, there is a method named poem that takes in an input without the bridge words (also symbol-free).

Example of “input.txt”

➔ *Seek to explore new and exciting synergies*

Output:

Print to the console the final string that is made. Don’t worry about the case of your output – you can just output every word in lowercase if you wish.

Example using “corpus.txt” and “input.txt”

➔ Seek to explore **strange** new **life** and exciting synergies

Weighted Graph:

What if more than one word followed the previous?

We create a bigram model where we keep track of the weights of each word.

Example:

Input 1:

To explore happy new worlds
To seek out new life and new civilizations

To explore strange new things

To explore happy new things

Input 2:

➔ *Seek to explore new and exciting synergies*

- *Notice there is two types of bridge words between ‘explore’ and ‘new’:*
  Explore -> happy -> new (2) Appears twice in the paragraph above
  Explore -> strange -> new (1) Appears once in the paragraph above.

As a result, there will be two edges between ‘explore’ and ‘new’

‘happy’ with weight 2

‘strange’ with weight 1

Return an output with the highest weight (i.e. weight 2):

Output:

➔ *Seek to explore **happy** new life and exciting synergies*

(Note: if multiple bridge words have the same weight, you can pick any one of them at random.)

Suggested Implementation

You are given 'Main.java' and 'GraphPoet.java'

Do not modify the name of the file or the given methods (although you can change the file paths in Main.java temporarily if necessary). You can choose to add any additional methods or classes to help solve your problem.

1. Constructing a Graph with Vertex.java

```
class Vertex<T>{
        private T name;
        private Map<T, Integer> edges; // T is a vertex, Integer is the weight
}
```

2. For every word in the corpus, create a vertex and a map of the word following it (bigram).
3. Take in the poem input and use the graph to find bridge words with the **max** weight.

Bonus (up to 10 points)

Web scrape a UT professor's web page, and attempt to generate a poem using their biography.

http://www.ece.utexas.edu/people/faculty/edison-thomaz

Example, scrape the above website:

Scraped Corpus:

Prof. Edison Thomaz is an Assistant Professor and Fellow of the Jack Kilby/Texas Instruments Endowed Faculty Fellowship in Computer Engineering in the Department of Electrical and Computer Engineering at The University of Texas at Austin.

Sample Input:

Prof Thomaz is a fantastic professor; engineering the highest levels of wearable sensors.

Sample Output:

Prof Edison Thomaz is a fantastic professor engineering at the highest levels of wearable sensors

(Note: your scraped corpus and input may have punctuation. Feel free to drop any non-alphabetic characters when making the graph and outputting the poem.)

You may use the library of your choice to perform the web scraping – one example is JSoup. You will also need to generate an input file, which you can name bonus_input.txt. To submit, include another class named Bonus in Bonus.java which contains a main method that runs your bonus code and outputs the poem. Please also submit a runnable JAR that runs this class and includes the library you used.

Submission Requirements

Name of zip file: Assignment_4_UTEID.zip. Make sure that the structure of the final ZIP file is as follows:

```
Assignment_4_UTEID.zip/
        assignment4/
                Main.java
                GraphPoet.java
                (any other files you create)
                …
```

Before submitting, please ensure that there is no output other than the poem itself.

Additional Considerations

- External Code: you are permitted to use any classes or interfaces within the java.lang, java.io, and java.util standard packages. You are specifically prohibited from making use of another student's code (including students who may have taken the class in previous semesters).

- Understandability: Comment your program so that its logic would be readily apparent to any software engineer who is familiar with standard data structures and algorithms.

- Re-use: Design your code so it is suitable for future adaptations and/or expansion.

- Warning: You may not acquire, from any source (e.g., another student or an internet site), a partial or complete solution to this problem. You may not show another student your solution to this assignment. You may not have another person (TA, current student, former student, tutor, friend, anyone) "walk you through" how to solve this assignment. Review the class policy on cheating from the syllabus.