CS:3330 Homework 6, Spring 2018

Due in class on Thu, March 8

---

1. Consider the 5-letter words in the text file `words.dat` that was posted for Homework 4. Define an undirected graph whose vertex set is exactly the set of words in this file. The edges of this graph connect every pair of words w and w' if and only if w and w' differ in exactly one letter. For example, the words `above` and `abode` would be connected by an edge, since they differ only in the 4th letter. Let us call this the *word graph*.

   Write a program that reads from `words.dat` and constructs an adjacency list representation of the word graph. To build an adjacency list representation that provides efficient access, first read the words from the file into an array W of size 5,757. Since the words appear in the file in sorted order, they can be placed in W in sorted order without much effort. Now build a second array, called A, such that for any word w, if w appears in slot i in array W, then slot i in the array A points to the list of the neighbors of w in the word graph.

   Now add two functions `neighbors` and `degree` to your program. Each function should take as arguments the arrays A and W and a word w. The `neighbors` function should return an array of all the neighbors of w in the word graph and the `degree` function should return the number of neighbors of w in the word graph. In order for these functions to be efficient, you should use binary search to figure out which slot in W the word w appears in.

   Now use your code to answer the following questions: (a) What is the highest degree in the word graph? (b) Which are the words that have this highest degree? (c) Which are the words that have degree 0 in the word graph? (d) What is the total number of edges in the word graph?

2. Now implement the functions `explore` and `DFS` that perform a depth-first search of a given graph. Building on these functions, write a new function that solves the ConnectedComponents problem. Recall that the output of the ConnectedComponents problem is a labeling of the vertices of the graph so that all vertices in the same connected component get the same label and vertices in different connected components get different labels.

   Now use your code to answer the following questions: (a) What is the size (i.e., number of vertices) of a largest connected component in the word graph? (b) What is the number of connected components in the word graph? (c) What is the number of size-2 connected components in the word graph?

3. A *separating vertex* is a vertex whose removal increases the number of connected components in the graph. For example, if the graph were connected (i.e., had just one connected component) then any vertex whose removal disconnects the graph (i.e., leads to two or more connected components) is a separating vertex.

   (a) To check your understanding of the definition of a separating vertex, find all separating vertices in the graph shown in Problem 3.31 (b) from the textbook.

   (b) DFS can used to identify separating vertices. To figure out how, first read Section 3.2.4 on the computation of $pre(v)$ and $post(v)$ for each vertex v. Then read the definition of $low(u)$ for a vertex u in Problem 3.31 (g). Modify the functions you wrote in Problem 2 to compute, $pre(u)$, $post(u)$, and $low(u)$ for all vertices u, in linear time

   (c) Once the values $pre(u)$, $post(u)$, and $low(u)$ for all vertices u are computed, it s pretty easy to use these values to identify separating vertices. Read the characterization of separating vertices in Problem 3.31 (e) and (f). (You don't have to solve these problems; just understand the statements you're being asked to show.) Then modify the `explore` and `DFS` functions to return an array of separating vertices.

(d) Finally, use your code to output all the separating vertices of the word graph.
