Contents

Changes to the Specification

24 Jul	words that are duplicates should be ignored (words may appear just once in any owl)
31 Jul	if there is more than one maximum owl, the list of owls must also be in alphabetic order

Assignment 2: Ordered Word Ladders

An *ordered word ladder* ('owl') is an alphabetically-ordered sequence of words where each word in the sequence differs from its predecessor by one action:

1. changing one letter, e.g. *barn→born*
2. adding or removing one letter, e.g. *band→brand* and *bran→ran*

The following are examples of word ladders of different length:

- *ban→bar→boar→boat→goat*, length 5
- *an→can→cane→dane→date→mate→mite→site→size*, length 9

Phase 1

At the heart of the assignment is a function that compares 2 arbitrary null-terminated strings and returns *true* if the strings satisfy one of the 2 conditions above, and *false* otherwise. This function has signature:

Toggle line numbers

```
1 bool differByOne(char *, char *)
```

Write such a function and of course test it.

Phase 2

Generate a graph that represents all the words in the input that *differ by one*. Each word is represented by a vertex in the graph, and vertices are adjacent if the corresponding words *differ by one*. For example, if a dictionary consists of the 7 words
`an ban bean ben hen mean men` then the graph that represents all *ordered word ladders* would be drawn as:

```
    0
    |
    1
  /   \
2 --- 3
|    / \
5---6---4
```

where the vertices 0..6 represent the 7 words in the given order. There are lots of ordered word ladders in this graph: for example, 0→1→2→5 representing *an→ban→bean→mean*. Any path between any two vertices in the graph is an *owl*, but notice that, although the edges are undirected, you can select vertices only in ascending order (the ladders must be in dictionary order).

In this phase of the assignment you are asked to create a graph for a dictionary, and simply print the graph.

- **Input** The words in the dictionary should be read from *stdin*. There will be *whitespace* between the words (i.e. spaces, tabs, newlines). You may assume that the words are in lower-case letters (so there are no capital letters, punctuation, hyphens ...) You may assume also the words are sorted in dictionary order. For example, an input file could consist of:

  ```
  an ban bean ben hen mean men
  ```

  You should use the *Graph ADT* from lectures to build your graph (I will be using the *Adjacency Matrix* version), and you are welcome to use any part of the *readGraph.c* program from lectures as well. You do not have to check the input for correctness (that is not what this assignment is about), so your program does not be have to handle 'bad' input (except for handling *whitespace*).
- **Output** Print the dictionary and the resulting graph (using *showGraph()* from the ADT). For the example above, the output of your program should look like:

  ```
  Dictionary
  0: an
  1: ban
  2: bean
  3: ben
  4: hen
  5: mean
  6: men
  Ordered Word Ladder Graph
  V=7, E=9
  <0 1>
  <1 0> <1 2> <1 3>
  <2 1> <2 3> <2 5>
  <3 1> <3 2> <3 4> <3 6>
  <4 3> <4 6>
  <5 2> <5 6>
  <6 3> <6 4> <6 5>
  ```

You may also assume in the assignment that the length of dictionary words is less than or equal to 20, and that there will not be more than 1000 nodes in the graph.

Phase 3

In this phase you need to find the length of the longest *owl* in the graph, which corresponds to finding the maximal path in the graph.

- You should first concentrate on dictionaries that have a single longest path, as in the dictionary above, which has one ladder of length 6, namely 0→1→2→3→4→6, and no other paths are longer. When you have determined the longest *owl*, print its length and the corresponding path as follows:

```
Maximum ladder length: 6
Maximal ladders:
 1: an -> ban -> bean -> ben -> hen -> men
```

  This output appears directly after the output above of course. Note that in general the maximal path may start at any vertex in the graph.
- Now address the problem of finding all the paths that have the longest length. All these paths should be output.

  For example, the input file `an at in it on` generates the following output:

```
 Dictionary
 0: an
 1: at
 2: in
 3: it
 4: on
 Ordered Word Ladder Graph
 V=5, E=6
 <0 1> <0 2> <0 4>
 <1 0> <1 3>
 <2 0> <2 3> <2 4>
 <3 1> <3 2>
 <4 0> <4 2>
 Longest ladder length: 3
 Longest ladders:
 1: an -> at -> it
 2: an -> in -> it
 3: an -> in -> on
```

  You see there are 3 *owls* here that have maximal length.

To test your program, you should create your own small dictionaries. If you want to see more of my examples see the links below.

1. 'case' example, 4 longest ladders
2. 'bear' example, 1 longest ladder
3. 'aaaa' example, 24 longest ladders

Assignment2 (last edited 2019-07-31 22:35:13 by AlbertNymeyer)