CS 300
DATA STRUCTURES

Homework 1

Assigned: **October 20, 2016**, Due: **October 31, 2016 at 23:55**.

PLEASE NOTE:

- SOLUTIONS HAVE TO BE YOUR OWN.
- NO COLLABORATION OR COOPERATION AMONG STUDENTS IS PERMITTED.
- 10% PENALTY WILL BE INCURRED FOR EACH DAY OF OVERTIME. SUBMISSIONS THAT ARE LATE MORE THAN 3 DAYS WILL NOT GET ANY CREDITS.
- SUBMISSIONS WILL BE MADE TO THE SUCourse SERVER. NO OTHER METHOD OF SUBMISSION WILL BE ACCEPTED.

In this homework, we are going to learn about how search engines such as Google do their searches really fast. These search engines search hundreds of millions web pages to see if they have the words that you have typed, and they can do this for thousands of users at a given time. In order to do these searches really fast, search engines such as Google do a lot of what we call *preprocessing* of the pages they search; that is, they transform the contents of a web page (which for the purposes of this homework, we will assume to consist of only strings) into a structure that can be searched very fast. Here is the basic idea:

- Let us assume we have millions of documents, $D_1, D_2, \ldots D_{19234678}, \ldots$, each consisting of hundreds or thousands of words. We identify each document by its number, e.g., 1, 2, ... 19234678, ...
- Since almost all the time these documents contain material in a human language, many words (such as *the*, or *geliyor*) most likely appear in many of the documents in a given language. Some words such as *the* or *bir* appear in perhaps all documents in a given language, while words such as *Llanfairpwllgwyngyllgogerychwyrndrobwll-llantysiliogogogoch*[1] or *esneyemeyense*[2] appear in relatively few documents.
- For each *unique* word, we construct a linked list of document numbers representing the set of documents that word appears in. We ignore details such as how many times a word appears in that document or where in the document it appears in, etc.

[1]Which is the name of a town in Wales, Great Britain.

[2]Which happens to be the longest palindromic Turkish word.

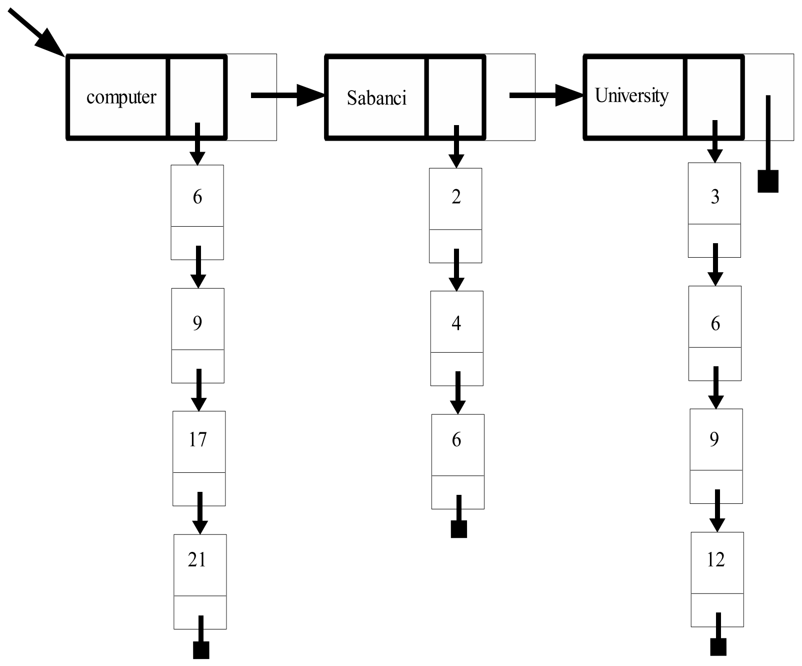

Figure 1: Conceptual layout of the sample database

For instance if the word `Sabanci` appears in documents D_2, D_4, D_6, we associate the list `(2, 4, 6)` with the string `Sabanci`. So, preprocessing involves finding ALL unique words AND creating a list of ALL documents numbers for the documents that contain that word somewhere.

- Once we finish preprocessing, we are ready to answer queries. Suppose we want to find all documents that contain the words `Sabanci` and `University`. We first identify the list of documents containing `Sabanci` and then identify the list of documents containing `University` and then *intersect* these, to get the list of documents containing both strings, and return the documents corresponding to these results as the answer. Of course, if you have more keywords, you keep on intersecting the result of the first intersection with the list for the third keyword, etc.

Here is how we expect you to attack this problem:

- Our main *database* of strings will be implemented as a list.[3] Each object we store in that list will be a pair of strings and their associated document lists. If in addition, `University` appears in documents D_3, D_6, D_9, D_{12}, and `bilgisayar` appears in documents D_6, D_9, D_{17}, D_{21}, our database will *conceptually* look like Figure 1.
- You should see that we use lists for two purposes: we use lists to store the list of documents associated with each string and we use lists to store the string-document

[3]As we will see later in the course, this is NOT the fastest method to implement this database.

list pairs. You may want to make sure that that list sorted based on the string values. It is also important that the list containing the document numbers associated with a string, is sorted in ascending order of the documents numbers.

- When you are given a query - a series of words - you search the database to locate the list nodes containing the words, and then retrieve their associated document lists and then intersect them.

Your task for this homework involves the following:

- You should supplement the basic list class with any additional functionality you feel you need. For instance, you may add methods for intersecting two lists, reversing a list, or inserting an element to its right place in a sorted list, etc. You should also define a class to handle the pairs of strings and lists of integers. Make sure you have reasonable documentation that lets us figure out what you are doing. You are free to use any source code we discussed in class.
- Your main program will read the database from a file called `docdb.txt`. Each line in this file will be a pair of string and a document number. For instance for the database above, the input file may look like.

  ```
  computer 21
  Sabanci 4
  Sabanci 6
  University 6
  University 12
  computer 17
  University 3
  computer 9
  computer 6
  Sabanci 2
  University 9
  ```

  Obviously any permutation of the lines above constitutes a valid database file for the data above. If some line is mistakenly repeated in the file you should ignore that line.
- You will then read query strings from the standard input. To simplify matters, each query will consist of a positive number and that many strings. For instance the query

  ```
  3 Sabanci University computer
  ```

  asks for all documents numbers containing the strings *Sabanci*, *University* and *computer*.

  If the number you read is 0, then the program should terminate without reading any further strings.

- The output for a query should consists of a line containing the query strings in the order given in the input, followed the number of documents found and then the matching document numbers in increasing order. There should be exactly 1 space between the strings and numbers in the output. For instance, for the query above the output will be

  ```
  Sabanci University computer 1 6
  ```

  indicating that there is only one document – document 6 – that has all the three strings.

  If however you have a query like

  ```
  1 microelectronics
  ```

  your output will

  ```
  microelectronics 0
  ```

  Indicating that there are 0 documents matching.

Your code should be submitted to WebCT at the deadline given on the first page. You should follow the following steps:

- name the folder containing your source files as *XXXX-NameLastname* where *XXXX* is your student number. For instance if your student number is 5432 and your name is *Ali Mehmetoğlu*, the folder will be named `5432-AliMehmetoglu`. Make sure you do NOT use any Turkish characters in the folder name. You should remove any folders containing executables (Debug or Release), since they take up too much space.
- Use the *Winzip* program to compress your folder to a compressed file named `5432-AliMehmetoglu.zip`. After you compress, please make sure it uncompresses properly.
- You then submit this compressed file in accordance with the deadlines above.

If the file you submitted is not labeled properly as above, if it does not uncompress, if the source code does not compile and if it does not work *for the input we will use*, as specified, we regretfully will have to give you 0 credit for the homework. So please make sure all these steps work properly.