STL for Competitive Programming and Software Development
Coding Test - April 2016

Victor Ciura - Technical Lead, Advanced Installer
Gabriel Diaconița - Senior Software Developer, Advanced Installer

Intro

Google Autocomplete

As you type in the browser search box, you can find information quickly by seeing search predictions that might be similar to the search terms you're typing.
The suggestions that Google offers all come from how people actually search. For example, type in the word "**cruise**" and you get suggestions like:

```
------------------------------------------------
Keyword: cruise
Suggested searches for keyword: "cruise"
   -> cruise line
   -> cruise ship
   -> carnival cruise
   -> caribbean cruise
   -> princess cruise
   -> disney cruise
   -> celebrity cruise
   -> norwegian cruise
   -> alaska cruise
   -> ship cruise
------------------------------------------------
```

These are all real searches that have been done by other people. **Popularity** is a factor in what Google shows. If lots of people who start typing in "**cruise**" then go onto type "**cruise line**" that can help make "**cruise line**" appear as a suggestion.

The Problem

You are given a keywords "database" in the form of a large text file (**keywords.db** - 136MB) containing **search terms (phrases)** used by people in the past (consider this an active search cache). Here is a small *fragment* from this text file:

```
------------------- keywords.db ------------------------
philips lcd 15
15 lcd cheap monitor
cheap 15 lcd monitor
dell e153fp 15 lcd midnight grey 36
lcd tv 15
samsung lcd 15
sony 15 lcd monitor
15 dvd lcd tv
15 inch lcd plasma monitors
-----------------------------------------------------------
```

You may assume that the text file contains only ASCII ***alphanumeric*** characters (English words). Keywords are separated by *space* or *CR/LF*.

The keywords database file is to be considered an **immutable** (read-only) snapshot of the current query **cache** to be used for **your own auto-suggestion engine for search terms**.

Each **line** in the file represents a search phrase used in the past. **For simplicity**, you should **ignore** the fact that the past search terms are organised on separate lines and consider the whole "database" as a *continuous chain of keywords*. A keyword is a sequence of non-whitespace characters (no normalization or input sanitization is needed).

Search phrase:

For simplicity, we shall define a search phrase as a pair of just **two consecutive keywords** in the query database. E.g. *"cruise line", "dell e153fp", "cruise ship", "samsung lcd", "norwegian cruise", "lcd cheap", "sony 15", "cheap monitor"*.

➽ Your <u>first task</u> is to load and **rank** the keywords database. That means ordering all search phrases (as defined above) according with their frequency in the cache (database).
Your program should be able to print to a file the **Top 1000** search phrases with their respective ranks (***occurrence*** frequency).
E.g. Top 10 search phrases from **`keywords.db`** are:

```
-----------------------------------------------------------
real estate # 43298
for sale # 38022
new york # 27302
how to # 25068
web site # 21073
las vegas # 19039
cell phone # 17657
of the # 15012
credit card # 14278
web hosting # 11037
-----------------------------------------------------------
```

➽ Your <u>second task</u> is to implement your own **auto-suggestion engine** for **10** related searches, based on top search phrases containing the input keyword.
See previous example with suggested searches for keyword: "*cruise*".
This operation (user inputs a new keyword and the engine auto-suggests related search phrases) should be repetitive during a program session and should be **super-fast**.
*** This <u>interactive mode</u> should be active only when the program receives a **`/search`** command-line switch.

➽ Provide an analysis of the **runtime** (steps) and **space** (memory usage) complexity in **Big-O** notation for all functions/code-blocks. The analysis should include the **average** and **worst-case** complexity along with a **brief explanation** of your reasoning. Write this information in **comments**; start them with: **// [SPACE COMPLEXITY]** or **// [RUNTIME COMPLEXITY]**