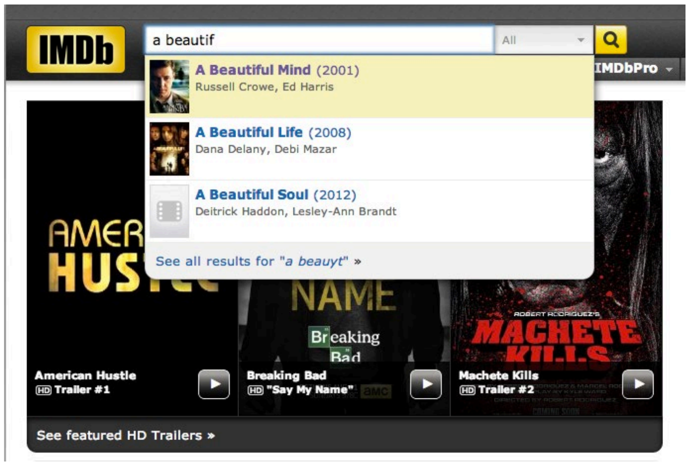
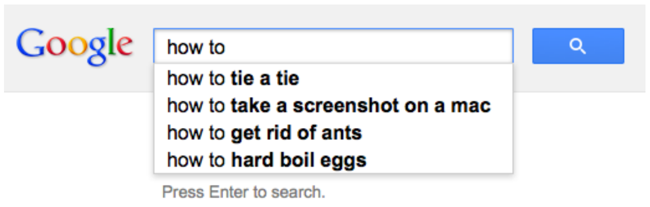
CS 2420 – Fall 2015
Program 5 –20 points   Priority Queues

Autocomplete

Write a program to implement *autocomplete* for a given set of *N* strings and nonnegative weights. That is, given a prefix, find all strings in the set that start with the prefix, in descending order of weight.

Autocomplete is an important feature of many modern applications. As the user types, the program predicts the complete *query* (typically a word or phrase) that the user intends to type. Autocomplete is most effective when there are a limited number of likely queries. For example, the Internet Movie Database uses it to display the names of movies as the user types; search engines use it to display suggestions as the user enters web search queries; cell phones use it to speed up text input.





In these examples, the application predicts how likely it is that the user is typing each query and presents to the user a list of the top-matching queries, in descending order of weight. These weights are determined by historical data, such as box office revenue for movies, frequencies of search queries from other Google users, or the typing history of a cell phone user. For the purposes of this assignment, you will have access to a set of all possible queries and associated weights (and these queries and weights will not change).

The performance of autocomplete functionality is critical in many systems.. According to one study, the application has only about 50*ms* to return a list of suggestions for it to be useful to the user. Moreover, in principle, it must perform this computation *for every keystroke typed into the search bar* and *for every user*!

In this assignment, you will implement autocomplete by using binary search to find the set of queries that start with a given prefix; and using a priority queue of the matching queries to produce the top k possibilities in descending order by weight.

Part 1: Create a max priority queue as a mergeable leftist heap. A max priority queue is a data structure that allows at least two operations: insert which does the obvious thing and deleteMax, that finds, returns, and removes the maximum element in the priority queue. We also want the heaps to implement a merge operation. Verify your priority queue works by creating a small test case which shows various combinations of the following operations:

- inserts into two different queues,
- prints the queue prettily (so the tree structure can be seen),
- merges the queues,
- deletes max

Create test cases which allows you (and the grader) to verify this is working.

Part 2: Perform autocomplete.

1. Read in the file and store it in an array of Terms.

2. For each target word (of length r), provide the top "count" matches using the following procedure

(a) Using a binary search on the array, find all words whose first r letters match that of the target word. You will need to implement two binary searches,

-
  ```
  //find the first location of key in a[] which matches r locations of key.
   int firstIndexOf(Term a[], string target, int r)
  ```

-
  ```
  //find the last location of key in a[] which matches r locations of key.
  int lastIndexOf(Term a[], string target, int r)
  ```

(b) Put all terms found in part (a) into a priority queue. Using weight as a priority, output the top "count" terms.

(c) Allow merging of queues

Input format

The first line of the SortedWords.txt file is the number of terms in the file. The remaining items are the word followed by its frequency.

User Interface

Allow the user to input (a) the prefix of a word (b) the number of outputs s/he wants to see (count)

Allow the user to say s/he wants the "and" of two words. *This will be accomplished by creating two priority queues and merging them.*

Feel free to add other user options if you want.

Output format The output will look something like:

```
Substring: acc  count=9

  according
  accept
  account
  access
  accident
  accuse
  accomplish
  accompany
  accurate
```

```
Substring: acc &all  count=11

    all
    allow
    according
    accept
    account
    access
    accident
    accuse
    accomplish
    ally
    accompany
```