CS2223: Algorithms | Assignment 3

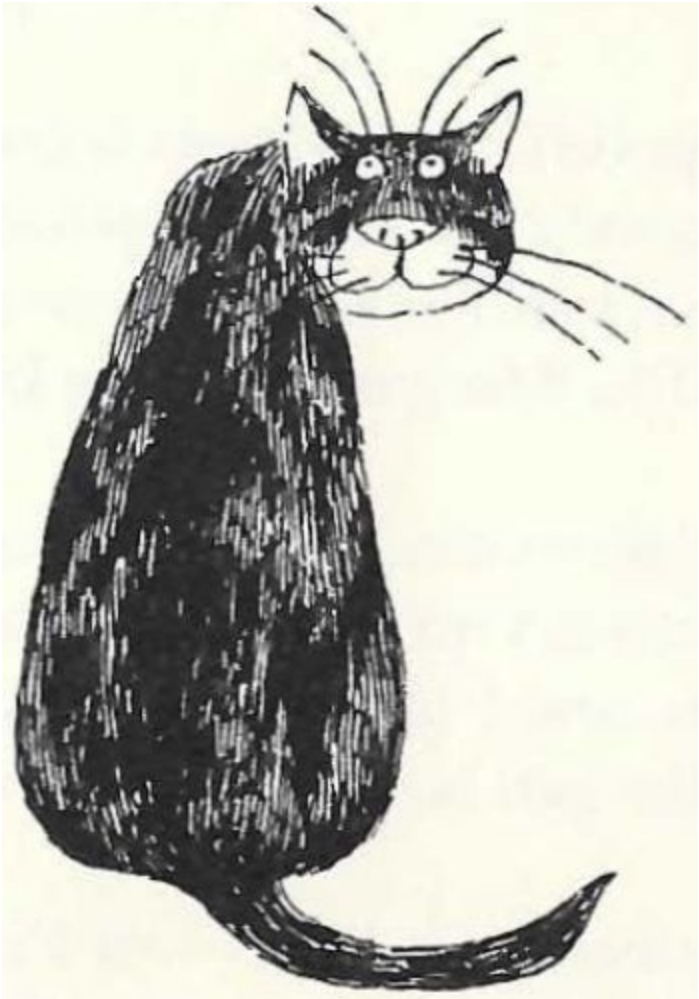
General Instructions

• Code vs. Pseudocode: Each question will state whether the deliverable is pseudocode or an executable program and code that the TA will run to give you a grade.

• You should make clear in your report:
– How to compile your program
– How to execute it, along with any arguments

• Submissions: Submit your work as a .Zip file through Canvas. All programs plus your written questions (which should be PDF files to ensure we can see them) should be zipped into a single archive when submitted. Please organize folders by question, e.g. Q1, Q2, etc.

Jellicle Cats are white and black,
Jellicle Cats are of moderate size;
Jellicles jump like a jumping-jack,
Jellicle Cats have moonlit eyes.



Q1: 😺🐈😺, HashTables, and Machine Learning (100 pts)

You are to assume the role of a Data Scientist in the not-too-distant future. You work for a small, independent production company that wants to put on a show of the Broadway classic *Cats.*

However, a large media conglomerate, Snidey, has bought the rights to the underlying poems that inspired the musical. Worse, due to advances in copyright law, you learn that Snidey plans to litigate and charge over any “unique” words in their corpus. The more unique the word is throughout the corpus, the more it will cost your production company to use it.

Your job is to mine this corpus of poems, and to build a rudimentary search engine that will allow the writers and business minds of your production company to explore the data and find words that need replacing. If successful, you plan to sell this software to other production companies.

Task Set 1 (Building):

1) Implement the LinearProbingHashST from Sedgewick.
2) Read in a series of plain text files (cats.zip), which we’ll call “Documents”
3) Study the simple, concise metric called TF-IDF (read beyond this link)
4) Build a “Term” class that will store: the word, the document it came from, the word’s frequency in the document, and (eventually) it’s tf-idf score.
5) Cycle through each document, and each document’s words.
    a) Build a hash table that stores <String, Term>, note that String is the key
    b) Increment the Term’s frequency if it’s already present
6) You should now have a hash table for every document.
7) Build another hash table of **all** words in **all** documents to get a measure of Document Frequency for all words.
8) Using the DocFreq hash table, cycle through and update the tf-idf measures for the individual doc hash tables.
9) You now have the elements for a rudimentary search engine.

Task Set 2 (Using):

10) Implement functions that demonstrate the following functionality:
    a) *search(Key)* For a given word, return the documents which contain the word, including the word’s frequencies and tf-idf scores in that document.
    b) *top10(Doc)* Return the top 10 terms by tf-idf scores for a given document.
11) Demonstrate these functions in your Main class with terms of your choosing.
    a) (Hint: most of these require iterating through the built tables.)

Task Set 3 (Analyzing)

12) Conduct an empirical evaluation targeting *search* and *construction*
    a) Obtain a sample of search items, by randomly choosing keys from the DocumentFrequency table. About 1/10 of the size will do.
    b) Using Stopwatch, test your search performance on these test sets, at least 10 times. Report these measurements.
    c) Repeat the Stopwatch operation for construction. No test set required.
13) Replace LinearProbingHashST with a Binary Search Tree for Symbol Tables (BST) (~398)
    a) Repeat the above measurements.
14) Produce a PDF that compares the two algorithms, and a brief explanation as to which is best for your company, and why.

For a Challenge:

- Enable sentence query search, e.g. “I like black and white cats”, using Cumulative Term Frequency, to return the most relevant documents to a sentence query.
- Obtain a larger text corpora (multiple text documents in a folder), to *really* test performance.

Output:

The output of this should include Java code TAs can inspect and run, as well as a PDF report that shows your analysis and tables or charts.