Google It! Program

Ali Alsemeyen, Jacob Everett, Devon Gates, Mitchell Koen

Project Description

Given a user-defined text file containing some number of documents, each consisting of an ID, author, title, and abstract delimited by ".I", ".A", ".T", and ".W" respectively, and a user-defined text file containing some number of stopwords delimited by newline characters, this program collects and organizes each piece of piece of data read in. It removes stopwords and punctuation and runs porter-stemming functions on each abstract.

Major Functionality Piece 1 (Jacob Everett)

The assignment for this section was to get and store information from a text document, categorize it based on tags within the document, and format and output what was read in.

The text file in our program is processed by a function called document::processDoc, which uses an ifstream object to read in the raw text line by line. Once the text file is read in, it is stored in a string. The string is then systematically searched and manipulated (str.find/str.erase/str.substr) based on where the tag denoting the next ID, author, title or abstract is found.

Once an abstract is processed and another ID position is found, the document just processed is push_backed onto a vector of document object pointers. If any tag's position ever equals a null value there are no more documents to be processed, and the first document's information is output to the user in a cout statement. Because each document is stored in the order it was read into the vector, the abstract located in docList[0] is the first document.

I chose to use a vector for storing the data in order to effectively store a potentially unknown number of documents that myself or my team members might need access to. Creating each document as an instance of the document class allowed me to separate each piece of data into its own variable, and using a vector to store each object allows for efficient iterations through the stored objects. In the future, I may choose to use a stringstream to read in the document rather than ifstream, but I didn't learn of it until halfway through my work on the second section. I believe that would have significantly decreased the number of lines I needed to process the file.

Major Functionality Piece 2 (Jacob Everett)

This section dealt with getting and storing the stopwords from a text document, then removing those words as well as all punctuation from each documents abstract and outputting the result of the first documents abstract.

The stopwords file is processed by a function called stopwords::processStops, which uses an ifstream object to read each character or word and store it in a vector called stopList while counting each word. This count is output to the user and the program returns to main where stopwords::docStop is called.

docStop copies the vector of document objects from the document class and stores it in a vector of document objects called noStopDocs. The text of each abstract is passed to the stopwords::isStopWord function, which takes a string and passes it through a stringstream, checking each word in the stream against the stopwords and returns true if a match is found. isStopWord also returns true for any solitary periods found. Whenever isStopWord returns true, that word is not stored in the cleaned abstract. If isStopWord returns false, meaning no matches were found, then the front and back of each word is checked for commas, slashes, and parentheses and erased if found. The resulting string is push_backed onto a vector called cleanAbs and cleanAbs[0] is output to the user. Each space in cleanAbs[0] is counted, and this number plus 1 (to account for last word) is output to the user before returning to main.

I chose to use a vector for storing this data for similar reasons to part one, but in retrospect, I should have used maps. I learned of some functionalities that maps have later in the project that I believe would have worked better. The implementation I went with does give me the advantage of being able to easily iterate through the data in each cleaned abstract using vector notation, such as when I iterated through each character of the cleaned abstract of the first document cleanAbs[0][i] in order to count the number of unique words.

Major Functionality Piece 3 (Devon Gates)

For this portion of the project, my job was to stem words using a porter-stemming algorithm. In order to make other parts of the project as easy as possible, I also stored all of the stemmed words in a vector accessible by a get function. I started off by doing it in C, but realized that memory management and allocation is much easier in C++.

All of my processing is done in C by converting the document to cstrings. I then convert the cstrings back into regular strings and push them into a 2D vector. The outermost layer in the vector represents the document number and the innermost represents each word in that document.

Major Functionality Piece 4 (Ali Alsemayen)

Outputting each word from the documents ".txt" in order to read word's term frequency, and inverse document frequency, and that's called TF-IDF, using the following equation:

$$tf * \log_e \frac{N}{n_t}$$

$\log_e \frac{N}{n_t}$ is used for the inverse document frequency. N is the is the number of the documents that's been read through.

n_t is used for the documents that's been collected in the current appearing word.

log_e is used for the natural log.

- Prompting the user in order to input the file name of the query.
- Removing the stop words, punctuation, and and perform porter-stemming.

Major Functionality Piece 5 (Devon Gates)

For part 5, my responsibility was to parse a user input by removing punctuation, removing stopwords, and stemming each word. I did this by using the infrastructure that I already implemented in the porter stemmer and infrastructure present in the tf-idf portion.

I stored each processed word in a vector. This way, I could remove stopwords in the middle of the vector, and the vector's built in functionality took care of allocating memory.

Major Functionality Piece 6 (Mitchell Koen)

For this portion of the project we calculated the cosine similarity between the query and each stemmed document. To calculate the cosine similarity between two documents, in this case the query and a document, we had to split up the equation (taken from instructions) into several parts $\frac{A \cdot B}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}}$

The numerator required the calculating of the Dot product between A and B. The denominator is calculated by finding the summation of every A^2 then taking the square root of the result. Afterwards the numerator would

be divided by the denominator and placed into a vector called vector 3, this vector would then be sorted from least to greatest and printed out.