Foster Clark

Patrick Hough

Gary Sidoti

Abe Arce

**Computer Algorithms Project II**

The goal of this assignment is to input a text file containing several paragraphs, parse the text files, remove stop words, stem the keywords, then index and query these key words. Removing stop words was the first goal. Since Ideone can only have a single file inputted the list of stop words, instead of being read in from a file, have been hardcoded in as a list. This list is then transferred to a hash map, which can easily be searched in constant time. As the document is parsed in, each word is compared to the Hashmap of stop words. If it is a stop word, the word is ignored and not added to the list of keywords that is being created. This is a simpler and faster method than actually removing the stop words from the full list, as there are no secondary iterations necessary.

Once the stop words are removed, we use the Porter algorithm to stem the words. The Porter algorithm works by assigning a “measure” of each word, determining the length of relevant characters and assigning rules for conversion. The rules for conversion check the word to see if it ends in one of the many stems. If the word does, the ending is removed, and sometimes replaced. For example, if the word ends with ATIONAL, it is replaced with ATE, giving the conversion relational 🡪 relate. In other simpler cases, like DOGS, only the last letter is removed, and nothing is put in the characters place. This general idea is done through five steps. Part one has three parts, checking for past tense words, adverbs and words ending with y. These are handled as special cases. Step two is more general, it does not focus on any specific suffixes, and does a general swap for certain endings. Step three is largely similar. Step four resembles three, however it checks to make sure that the word with the longest suffix is used. Finally step five mostly cleans up, by checking if the word ends in the character ‘e’ and pops the character off the string if it does.

Once all words have been parsed, stop words removed, and the remaining key words stemmed, they have to be indexed. Since Ideone does not accept multiple documents, we are indexing by paragraph instead of by document. We accomplished this through node manipulation. As each word is read, the paragraphs are counted. We first create a list of paragraphs. Each paragraph consists of the stemmed keywords placed into nodes. Each of these nodes contains a list of ints that catalogs where each word appears. So essentially a 3D array is created, which is easy to iterate through. Once this is created, we iterate though these lists and output each keyword, as well as where it appears in the documentation.

Finally the searching is done in a very similar fashion to the indexing. Once again, due to Ideone’s limitations, instead of cataloging documents, we will return what paragraphs the words appear in. This is done by iterating through the Paragraph list and counting where the query appears, and how many times it does so. The result is then outputted to the user.

By using the hash table to check for stop words, and a matrix to catalog paragraphs and key words, we have optimized the run time the best we are able to. The Porter algorithm shortens and simplifies words so there are less keywords to search for. The less iteration we have to perform, especially with larger documents, the more time we can save.