CS454 Assignment 1: Building an Indexer
Assigned: Saturday, October 8, 2005
Due: 12:00 PM, Tues, October 25, 2005

1 Project Description

An Indexer is a core part of any search engine; it creates an index (inverted file) of terms from a repository, a set of web documents (also known as a crawl), which has been generated by a spider (or crawler) from the Web.

Your job is to write an Indexer. Your code should record every term, along with its position(s) within the document, from every page in a repository. Such an index could be queried by a search engine in order to find hits for search terms; indeed, later parts of the assignment will deal with different search-engine components. For simplicity, the interface to the repository of documents has been written for you.

2 Project Objectives

By the end of this project you should understand what information is needed to build an index and what files and data structures are needed to store that information for later retrieval. This project should not entail a massive amount of programming.

For this project, you will index all the words, composed soley of alphabetical characters, in document bodies. You should convert all words into lowercase. Word positions should be represented by integers, with 0 being the first position. We will provide your program with a list of "stop words," which should not be indexed because they are so common that they bear little meaning.

As we discussed in class, your indexer must not try to use main memory for all data structures. You may assume that a hash table mapping the lexicon into pointers can fit in memory, but not the occurrence list which is being pointed to.

3 Report

After you have written your indexer, write a short paper about it. Describe the data structures you chose to use, and the format of your saved index. Make sure your code, your code comments, and your paper are consistent; they will be read and considered together. Each method and class should have a header comment describing what's going on. An independent reader should be able to understand what your code is trying to do and why.

Some questions you might want to think about answering:

- How much memory does your indexer use? What if your machine had even less — how would you modify it to keep working?

- How many disk accesses does your indexer use? Can you think of a way to improve it (or argue that this would be impossible).
- Is there a significant effect when you add more stop words to your indexer? What could be a downside to adding more words?
- (Optional) How would you modify your code if the lexicon couldn't fit into main memory?
- (Optional) How would you modify your indexer to do updates — i.e. if the spider is running continuously and you had a new repository, which you needed to merge into the indexer.?

Your paper should also include some brief experimental results. Some things to experiment with could be indexing other reasonable strings, besides those that only contain alphabetical characters, or a different list of stop words, and their effects on your index size. Or explore the bulletted questions above.

4 Getting Started

Before we can write the indexer, we need to figure out how to run the rest of the engine.

4.1 The CSE454 Project Framework

You can invoke a command-line version of the CSE454 project framework with the following command: `/projects/instr/cse454-05au/assignment1/bin/cse454`

Try running this program, and you'll see this:

```
Usage: cse454 COMMAND
where COMMAND is one of:
  index         Index the CSE454 crawl, using your indexer code.
Commands print help when invoked w/o parameters.
More commands will be available in future assignments.
```

It's a little silly to have only one option here, but there will be more later on in the course. So now run the program with the parameter "index" and you will see:

`Usage: cse454 index [-crawlsize small, med] -indexclass your.index.class [optional arguments for your indexer]`

This command allows you to index all the documents in one of two document sets. (Right now the small set consists of about 3000 docs, while the medium consists of a larger set of docs, TBA.)

Unfortunately, the engine can't find its indexer right away. You need to tell it how to find one. After you compile the class `edu.washington.cse454.StudentIndexer`, you can tell the

engine about it by setting the CSE454_CLASSPATH environment variable. Set this value to the directory that contains your `StudentIndexer` class, and you'll be ready to go. [1]

If we run `cse454 index -indexclass edu.washington.cse454.StudentIndexer` we'll see output like this:

```
Creating edu.washington.cse454.StudentIndexer
051005 151147 loading file:/projects/.instr/cse454-05au/assignment1/conf/nutch-default.xml
Initializing indexer... done.
Indexing pages... done.
Saving index... done (saved to /).
```

You can safely ignore the first line. The second line indicates that the specified indexer will be initialized. The third line indicates that, once initialized, the indexer will be given the document set to index. The last line indicates the file where the indexer saves its index (this is for you to turn in).

4.2 Interfaces

You can find skeleton code for `edu.washington.cse454.StudentIndexer` by looking here:
`/projects/instr/cse454-05au/assignment1/samples/StudentIndexer.java`

This file is reproduced in Appendix A.

You should copy this file to your work area and use it as a starting point.

Take a look at all the relevant interfaces here:
`/projects/instr/cse454-05au/assignment1/reference`

4.2.1 IIndexer

The `StudentIndexer.java` class implements the following interface:

```
public interface IIndexer {
    public void init(Set stopwords, Pattern wordseparators, String args[]);
    public void indexPages(Document.Enumerator pages) throws IOException;
    public File saveIndex() throws IOException;
    public TermEnumerator loadIndex(File index) throws FileNotFoundException,
            IOException;
    public TermEnumerator getEnumerator();
```

[1] Well, that's almost true. Remember that Java places classes in directories that are generated from the class package name. So Java expects that the `edu.washington.cse454.StudentIndexer` class will appear in a file called `StudentIndexer.class` in a directory hierarchy named `edu/washington/cse454/`. If your compiler is outputting to a directory called `classes`, and you have your compiled class at `classes/edu/washington/cse454/StudentIndexer.class`, the CSE454_CLASSPATH variable should be set to `classes`. The Java runtime will look in the right subdirectory.

```
    public FileList getHits(String term);

    interface TermEnumerator {
        public boolean hasNext();
        public FileList next();
    }

    /**
     * FileList is a data structure that can be populated with a term and a
     * list of DocOccurrence objects, which represent that term's location
     * in the crawl.
     */
    class FileList {
        public String term;

        /*
         * This should be a list of DocOccurrence objects, each representing one
         * document that contains the term
         */
        public List docOccurrences;

        /**
         * DocOccurrence keeps a list of positions within a single document.
         */
        public class DocOccurrence {
            public int docId;

            /* The total occurrences of the term within this document */
            public int freq;

            /*
             * This should be a list of Integer positions within the
             * document where the term appears.
             */
            public List occurrences;
        }
    }
}
```

When the CSE454 framework starts up, it will call the `IIndexer.init()` just once. The passed-in `Set` object contains the stop words you should not index. The `Pattern` object defines the characters that separate words to be indexed. The `String[]` object contains the additional arguments that you may wish to pass to your program, via the cse454 command.

The framework will call `indexPages()` with a `Document.Enumerator` object over all `Document` objects representing the documents in the crawl. `saveIndex()` will be called afterwards to save the index to a file.

4.2.2 Document

Thes aforementioned `Document` class contains data and metadata about a document in the crawl. `Document.Enumerator` enumerates over a collection of `Document` objects.

```
public interface Document {
    public String getURL();
    public String getTitle();
    public String getText();
    public int docId();

    interface Enumerator {
        public Document nextDocument() throws IOException;
        public boolean hasNext();
    }
}
```

5 What to Hand In

- Your implementation of StudentIndexer, with extensive comments. Be sure your indexing function is clear and easy to understand.
- Your saved index of the medium sized crawl that the TA can load with your `loadIndex()` method.
- A description of your indexer, no more than 4 pages in length. Describe the file and data structures you used and why you chose to use them.

  Be sure your description of your ranker and your choices behind it match your code and your code comments.

We will supply details on the handin process via the class mailing list.

6 Grading

The grade for this assignment will be computed as follows:

- Clear and sensible file structure, data structure, and algorithm choice, cleanly implemented and described, 40%
- Clear and convincing writeup, 40%
- Correctness based on comparison with TA algorithm, 20%

Late assignments will be penalized as described in the class policies at http://www.cs.washington.edu/education/courses/454/05au/policies.html.

7 Groups and Collaboration

You will work alone on this first project. (Later projects will be collaborative.) Discussions among different students are allowed, subject to the Gilligan's Island rule and other important directives listed in the class policies.

8 Appendix A

Here is the skeleton code found at
`/projects/cse/courses/cse454/05au/assignment1/samples/StudentIndexer.java`:

```
package edu.washington.cse454;

import java.io.File;

/**************************************************************
 * StudentIndexer is an unfinished class you must implement for
 * your indexer to work correctly.  Most of the indexer
 * code already exists; you 'only' need to write the
 * code that records, saves, and loads the occurrences of all
 * words in the crawl.
 *
 * This class *must* implement the IIndexer interface, or else
 * it will not work with the CSE454 framework code.
 *
 * For more info, see the course webpage at
 * http://www.cs.washington.edu/education/courses/454/
 *************************************************************/
public class StudentIndexer implements IIndexer {
    /**
     * The init() method is called immediately after the class
     * is created, and is called only once.
     *
     * It provides your instance with a Set of stopwords, the
     * precompiled Pattern that defines the separation between words to
     * be indexed, and any additional arguments you pass to the cse454
     * command.
     *
     * @see Pattern#split(java.lang.CharSequence)
     */
    public void init(Set stopwords, Pattern wordseparators, String[] args) {
    }

    /**
     * The indexPages() method is called after initialization with a
```

```
 * Document.Enumerator object that allows you to get the document id,
 * title, and text (stripped of HTML) of every page in the crawl. For
 * this project, you should not index the title (it complicates the
 * notion of word position), but it might be useful to know for various
 * reasons.
 */
public void indexPages(Document.Enumerator pages) {
}

/**
 * The saveIndex() method is called after indexPages() finishes. This
 * method should save the entire index into a file (or directory, if you
 * use multiple files) that you can submit along with your code and
 * write-up.
 */
public File saveIndex() {
    return File.listRoots()[0];
}

/**
 * The loadIndex() method should load an index from a file or directory
 * that was created by your saveIndex() method. This method will be called
 * in order to compare your index with the TA's for grading purposes. It
 * must return a TermEnumerator object that enumerates all terms found in
 * the index.
 */
public IIndexer.TermEnumerator loadIndex(File index) {
    return null;
}

/**
 * The StudentIndexEnumerator is an implementation of TermEnumerator.
 * You are free to rename it  It provides an interface over your
 * index that allows for the enumeration of all words in the crawl and
 * their document positions.
 *
 * @see #getEnumerator()
 */
class StudentIndexEnumerator implements IIndexer.TermEnumerator {
    public boolean hasNext() {
        return false;
    }

    /**
     * The next() method should return a populated FileList object.
     */
```

```
        public FileList next() {
            return null;
        }
    }

    /**
     * The getHits() method should return a FileList object populated with
     * all occurrences of term in the index, or null if term is not present
     * in the index.
     */
    public FileList getHits(String term) {
        return null;
    }

    /**
     * The getEnumerator() method must return a TermEnumerator
     * object that enumerates all terms found in the index.
     */
    public TermEnumerator getEnumerator() {
        return new StudentIndexEnumerator();
    }
}
```