Pirex

Overview of Information Retrieval Systems

1.1 Information Retrieval

Information retrieval (IR) is the study and application of means to store, search, retrieve and disseminate data from large textual databases. IR is the basis for web search engines like Google, tools for searching the contents of files (like Spotlight on a Mac), email search tools, and tools to search for products at online stores like Amazon. IR technology is also used in services that route material (like news articles) to people who have expressed interest in certain topics, in systems that help analyze texts by clustering or categorizing them, in systems that summarize large texts, and even in some expert systems and question answering systems.

This semester you will form teams and do a project that uses IR techniques, thus giving you an opportunity to learn a little bit about this important area of Computer Science.

1.2 IR System Structure

A typical IR system is built around a data structure called in inverted index. Texts contains words, each with positions in the text. An **inverted index** (or **inverted file** or **postings file**), is a map from words to texts and text positions. For example, suppose in a collection of books the word *copacetic* occurs in the third book, the eighth book, and the 73rd book. In the third book it occurs as the 742nd and 9,923rd word, in the eighth book at the 2,723rd word, and in the 73rd book as the 4,361st, 4,420th, and 5,345th words. Then the inverted index for this collection would have an entry like the following:

copacetic	<3, {742, 9923}>, <8,{2723}>, <73, {4361, 4420, 5345}>

The word is called in **index term**. A document identifier and a set of term locations is a **posting**. Hence an entry in an inverted index is an index term and a list of postings. Most of the words in every text in a collection are included as index terms in the inverted index.

When users do a search, they form queries from words, called **search terms**. A **query** is a sequence of search terms possibly including operators specifying search parameters. Each search term is looked up in the inverted index and the postings for the corresponding index terms are combined and used to fetch documents from a document store to provide a set of results, a **retrieved set,** displayed to the user.

We thus see that IR systems have two major parts: an *administrative* part that takes texts as input and builds inverted indexes and document stores, and a *search engine* that takes queries as input, matches them against the inverted index, fetches texts from a document store, and then displays retrieved sets to users.

Project SRS v1-1

1. Introduction

Pirex (personal information retrieval experimental system) is an information retrieval system that individuals can use to investigate their own texts.

1.1 Product Vision

Pirex allows users to input arbitrary plain-text documents. It indexes and stores input texts permanently in the file system so they can be searched later. It allows users to enter queries to retrieve paragraphs from stored texts matching the queries, and displays retrieved paragraphs to users. Pirex has a graphical user interface for all its functions. Pirex may include various auxiliary features to enhance search capabilities and results investigation.

1.2 Project Scope

The Pirex project is time-boxed between Aug 27th and Dec 14th, 2018, with a team of five to seven developers using Scrum.

1.3 Stakeholders

Project stakeholders include the following.

- The developers
- The class instructors (customers)
- The users

1.4 Design and Implementation Constraints

1. Pirex must be written in Java.

1.1 Pirex must have a GUI implemented with Java Swing.

2. Pirex must be developed using Eclipse.

2.1 The Pirex Eclipse project must use Checkstyle with the project-supplied configuration file.

2.2 Pirex must conform to project coding guidelines as indicated by not exhibiting any Checkstyle violations.

3. Pirex must have a complete set of JUnit tests.

3.1 The non-GUI portions of Pirex must have JUnit tests that achieve greater than 90% statement coverage.

4. All Pirex source code must be under source code version control.

4.1 The version control system used must be Git.

4.2 The instructor must have access to the Git repository for the project.

2. Functional Requirements

2.1 Product Behavior

1. Pirex must have features for *administrators* and *users*.

2. Administrator must be able to load text files for later search and retrieval.

2.1 The administrator must be able to select text files on the computer where Pirex is running.

2.1.1 The text in a text file is referred to as an *opus*.

2.2 Each opus must have a title, an author, and an ordinal number identifier.

2.2.1 If an opus has no title, the opus title must be recorded as "None."

2.2.2 If the title is extracted automatically from the opus, then the user must be able to edit the title.

2.2.3 If the opus has no author, the opus author must be recorded as "Anonymous."

2.2.4 If the author is extracted automatically from the opus, then the user must be able to edit the author.

2.3 The opus must be broken into paragraphs, which will be the textual units retrieved (*documents*).

2.3.1 Each document must be associated with its opus title.

2.3.2 Each document must be associated with its opus author.

2.3.3 Each document must be associated with its ordinal number (that is, by the numbering of paragraphs from the start of the opus).

2.3.4 Each document must be indexed and the results stored in the file system for later use.

2.3.5 Each document must be stored in the file system for later retrieval.

2.4 At the conclusion of loading a text file, Pirex must summarize the result of the loading operation.

2.4.1 The loading summary must list the source file, opus title, opus author, opus ordinal number, number of documents extracted from the opus, the number of new index terms generated during the indexing operation, the number of new postings generated during the indexing operation, the total number of terms in the index, and the total number of postings in the index.

2.5 Pirex must be able to load at least 10 text files.

3 The administrator may be able to remove an opus.

3.1 Every document in a removed opus must be removed from the file system.

3.2 Every reference to deleted documents must be removed from the inverted index and the changes stored in the file system.

3.3 Every index terms in the inverted index with no postings as a result of deleting an opus must be removed from the inverted index and the changes stored in the file system.

3.4 At the conclusion of an opus removal operation, Pirex must summarize the result of the operation.

3.4.1 The opus removal operation summary must list the opus title, opus, opus ordinal number, number of documents removed, number of index terms whose postings were altered, and the number of index terms removed completely from the inverted index.

4. The administrator must be able to view an opus summary.

4.1 The opus summary must list all the opuses currently saved in Pirex.

4.2 For each opus, the summary must list the source text file name, opus title, opus author, opus ordinal number, and the number of documents in the opus.

4.3 The opus summary text must list the total number of index terms and the total number of postings.

5. Users must be able to enter *basic queries* to retrieve documents.

5.1 A basic query must allow users to enter from one to 100 *search terms*.

5.2 Pirex must retrieve all documents indexed by all search terms in a basic query.

5.2.1 A basic query must be broken into search terms using the same rules as are used to break a text into terms.

5.2.2 All stop words (except those used with the adjacency operator) must be removed from a basic query before any stemming and matching occur.

5.2.3 If stemming is used, then all search terms must be stemmed using the same stemming algorithm as for indexing before term matching is done.

6. User may be able to enter *advanced queries* that use various operators.

6.1 The *adjacency operator* is the caret (^) character.

6.1.1 An adjacency expression is a string of the form t_1^t_2^ … ^t_n where t_1 through t_n are search terms. Stop words may be used as terms in adjacency expressions.

6.1.2 An *adjacency expression* t_1^t_2^ … ^t_n must match all and only documents in which the terms t_1 through t_n occur in order separated only by whitespace characters.

6.2 The *not operator* is the tilde (~) character.

6.2.1 A *not expression* is a string of the form ~t where t is a search term.

6.2.2 A not expression ~t must match all and only documents in which term t does not occur. (Note: a not expression cannot be part of an adjacency expression).

6.3 An advanced query is a string of the form q_1 q_2 … q_n where each q_i is either a search term, an adjacency expression, or a not expression.

7. The set documents retrieved from a query are *search results* or the *retrieved set*.

7.1 When a query is processed the number of documents retrieved must be displayed.

7.2 Search results must be displayed to users initially in a *short form*.

7.2.1 The short form document display must include the opus title, opus author, the document ordinal number, and the first line of the document.

7.3 Users must be able to select a document from the short form display and expand it to a *long form* display.

7.3.1 The long form document display must show the entire document.

7.3.2 Users may be able to dismiss a long form display.

8. Users must be able to clear a query.

8.1 Clearing a query must also clear all retrieved documents.

8.2 Clearing a query must also remove the display of the number of documents retrieved.

9. Users must be able to enter a sequence of queries.

9.1 When a new query is entered, the results of the previous query must be replaced with the results of the current query.

10. Pirex must break text into terms to use as index terms when loading an opus.

10.1 *Whitespace* characters are space, horizontal tab, vertical tab, line feed, carriage return, newline, and backspace.

10.2 *Punctuation* is all non-whitespace, non-letter and non-digit characters.

10.2.1 Dashes used to form compound words (as in *twenty-six*, for example), must be treated as letters. Dashes used in other contexts must be treated as punctuation.

10.2.2 Single quotes used as apostrophes (as in *isn't*, for example), must be treated as letters. Single quotation marks in other contexts must be treated as punctuation.

10.2.3 Decimal points used in numbers (as in 3.141592) must be treated as digits. Dots occurring in other contexts must be treated as punctuation.

10.3 Pirex must use whitespace and punctuation as delimiters to distinguish words and numbers.

10.3.1 Words are sequences of numbers and digits with at least one letter.

10.3.2 Numbers are sequences of digits.

10.4 Pirex must use all words and numbers in a document as index terms.

10.5 Pirex must convert all letters in words extracted from a text to lowercase for use as index terms.

10.6 Pirex must not include the following words as index terms (the *stop list*): a, an, and, are, but, did, do, does, for, had, has, is, it, its, of, or, that, the, this, to, were, which, with.

10.6.1 Pirex may use a larger stop list by adding one or more of the words in the file stopList.txt (found on Canvas) to the list above.

10.7 Pirex may use the Porter stemming algorithm to stem index terms.

11. Pirex must perform the same processing on basic queries as it does on index terms to produce search terms.

12. Pirex must store data in the file system in a designated directory.

12.1 The Pirex data store directory must be the one specified in an environment variable named PIREX_STORE.

12.2 If the PIREX_STORE environment variable is not set, or it designates a directory that is not usable, then the Pirex data store must be the directory pirexData in the user's current working directory.

12.3 The Pirex data store directory must be created if it does not exist.

13. If Pirex encounters errors that it cannot recover from, it must write a message containing all available data about the failure to the standard error output and halt.

13.1 If the Pirex data store directory cannot be created, or if it exists but cannot be used, then Pirex must write an error message and halt.

13.2 If Pirex is encounters an IO error when loading or saving its data store, then it must write an error message and halt.

14. If Pirex encounters a read error in the midst of reading a text file that it is loading, then it must treat that file as it were completely processed but write an error message in the summary of the results fo the loading operation.

14.1 The error message written in the summary of the loading operation must indicate that the file was not completely processed and must indicate the error that occurred, and if possible the point in the file at which it occurred.

2.2 User Interfaces

See the separate UI design document (IDD_v1.pdf).

2.3 System Interfaces

1. Pirex must run on any machine that supports Java.

2.4 Data Requirements

1. Pirex must be able to load plain text books downloaded from Project Gutenberg (www.gutenberg.org).

1.1 Users must specify to Pirex that a file is a Project Gutenberg file before processing it (in other words, Pirex need not detect Project Gutenberg files without any user guidance).

1.2 Pirex must extract the author and title from Project Gutenberg books automatically.

1.3 Pirex must not index and store the head and tail portions of Project Gutenberg books (the portions delimited by the ***START and ***END markers).

2. Pirex may be able to load HTML documents.

3. Pirex may be able to load Rich Text Format (rtf) documents.

3. Non-Functional Requirements

3.1 Pirex must run without failure with a document collection from at least four text files and execution of at least ten basic queries.

3.2 Pirex must load a text file with 100,000 words into a collection with at least four text files in less than four seconds.

3.3 Pirex must retrieve documents for a query of two terms against a document collection from four text files of 100,000 words each in less than one second.

4. Other Requirements

None.

5. Glossary

adjacency expression—a string of the form t_1^t_2^ … ^t_n where t_1 through t_n are search terms; stop words may be used as terms in adjacency expressions.

adjacency operator—the caret (^) character.

advanced query—a string of the form $q_1\ q_2\ \ldots\ q_n$ where each q_i is either a search term, an adjacency expression, or a not expression.

basic query—a query composed of one or more search terms with no operators.

digit—the characters '0' to '9'.

document—a text unit retrieved in an information retrieval system; in Pirex, a document is a paragraph from an opus.

document collection—all the documents stored in an information retrieval system.

index term—a sequence of characters stored in an inverted index that is used to retrieve documents.

indexing—the process of analyzing a text to generate index terms.

inverted index—a data structure that maps index terms to postings.

letter—the characters 'a' to 'z' and 'A' to 'Z'.

not expression—a string of the form ~t where t is a search term.

not operator—the tilde (~) character.

number—a sequence of digits possibly including a decimal point; the decimal point must be both preceded and succeeded by at least one digit.

opus—the text in a text file.

posting—a document identifier and possibly other data about an index term in a document.

punctuation—a characters that is not a letter, digit, or whitespace.

query—a sequence of characters used to retrieve items.

retrieved set—the set documents retrieved by processing a query.

search results—a retrieved set.

search term—a sequence of characters matched against index terms to retrieve documents.

stemming—the process of removing the ends of words (including but not limited to suffixes) to create a common index term. For example, the words "draw," "draws," "drawable," and "drawing" can be reduced to the index term "draw."

stop list—a list of terms not used as index or search terms.

stop word—a term in a stop list.

text—any sequence of characters, usually stored in a file.

whitespace—space, horizontal tab, vertical tab, line feed, carriage return, newline, or backspace.

word—a sequence of letters and digits containing at least one letter; a word may contain an apostrophe or a dash if it is both preceded and succeeded by a letter.