ADDING A SOURCE CODE SEARCHING CAPABILITY TO YIOOP

CS297 REPORT

Submitted to

Dr. Chris Pollett

By

Snigdha Rao Parvatneni

1. INTRODUCTION

The aim of the CS297 project is to explore and learn important components to incorporate a source code searching feature to Yioop, a PHP-based search engine. The source code searching feature will allow users of Yioop to search Java and Python source codes from open source code repositories like GIT. All that a user would need to do is to type or paste a Java or Python code snippet in the search bar of Yioop and click the search button. This action will return the set of relevant source code files to the user. A user could then go over the search results to find the implementation for which they are searching.

Deliverables in the CS297 project gave me an opportunity to learn PHP, to understand GIT, and to implement the Naïve Bayes classifier. These deliverables gave me a fair idea of how to implement the source code searching feature in Yioop for the CS298 project.

This report includes details of all the deliverables of CS297 project. Before starting with the actual deliverables, preliminary activity included installing and configuring Yioop and writing a patch to support Bengali language in Yioop. The first deliverable was about learning how Sourcerer and Google Code Search work. The second deliverable was to implement a Naïve Bayes classifier in Java to recognize Java and Python source code files. The third deliverable had two parts: Implementing Naïve Bayes classifier in PHP to get familiar with PHP and enabling Yioop to process Java and Python source code. The fourth deliverable was reproducing the effects of GIT clone via cURL calls without using any external libraries. The last deliverable was the CS297 project report.

In this report each deliverable is explained under appropriate section headers. At the end, the report has the conclusion section to discuss the learning from the project followed by the reference section including all references used to achieve the goals of the project.

2. OVERVIEW OF DELIVERABLES

2.1 Deliverable-1

The purpose of this deliverable is to analyze the various steps involved in search operations of existing source code search engines. Sourcerer and Google Code Search were selected for the study. After an internet search, I found papers and articles written for Sourcerer and Google Code Search. Important points from secondary research of Sourcerer and Google Code Search are presented.

2.1.1 Sourcerer

As mentioned by Bajracharya et al. (2006) in the paper "*Sourcerer: A search engine for open source code*", Sourcerer is a search engine developed for academic purposes and it is used for searching open source code files in Java.

Sourcerer performs two important activities: searching and ranking. As mentioned by Bajracharya et al. (2006), Sourcerer extracts structural information from the source code and uses it for finding search results and ranking the search results.

The main aim of extracting structural information is to extend the Sourcerer's capability beyond traditional keyword-based search. Bajracharya et al. (2006), categorizes the source code search into three broad categories: Implementation search, usability search, and structural characteristic search. After this, Bajracharya et al. (2006) explain the Sourcerer's architecture, and describe various components of the architecture. Sourcerer's architecture helped me in understanding the overall infrastructure of the search engine and also explained to me the roles played by different components in searching and ranking activities.

Bajracharya et al. (2006) provide a detailed explanation about how significant features are extracted from source code files and how these extracted features are stored. According to Bajracharya et al. (2006), source code features are entities and relations. Additionally, keywords and fingerprints are used for quickly retrieving search results. As mentioned by Bajracharya et al. (2006), extracted features are stored in the relational database and in the form of fingerprints. Fingerprints represent the quantifiable features linked to the entities (Bajracharya et al. 2006). Extracted source code features enable Sourcerer to perform structure-based searches.

The next section of Bajracharya et al. (2006) explains how Sourcerer ranks the retrieved search results. As mentioned by Bajracharya et al. (2006), while ranking, Sourcerer treats codes as text and makes use of the TF-IDF heuristic. However, Bajracharya et al. (2006) explains that source code files are different from free text, so their structural information can be used along with TF-IDF to improve the Sourcerer's ranking scheme. The researchers suggested using three different heuristics in Sourcerer for ranking the search results. As explained by Bajracharya et al. (2006), the first heuristic uses a text-based approach, the second heuristic takes advantage of a structure-based approach, and third heuristic incorporates a graph-based approach. Finally, Bajracharya et al. (2006) experimented with all these three heuristics in Sourcerer. Comparing the recall of the first ten and twenty hits, Bajracharya et al. (2006) found that the combination of all the three heuristics performs the best.

2.1.2 Google Code Search

As mentioned by Cox (2012) in his article "*How Google Code Search Worked*," Google Code search is no longer an active project. According to Cox (2012) Google Code Search uses Google's famous concept of document indexing and retrieving search results.

In this article, Cox (2012) suggests that Google Code Search uses a regular expression search over an inverted index. Cox (2012) explains that Google Code Search uses an inverted index to find the plausible search results. The inverted index is built by trigrams because search operations do not perform well over the word boundaries. Moreover, n = 3 is selected for character-n grams instead of selecting $n = 2$ or $n = 4$ because if n is selected as two then there will be too few distinct 2-grams, and if n is selected as four there will be too many distinct 4-grams (Cox, 2012).

As explained by Cox (2012), a search string is treated as a regular expression, which is chunked into trigrams. All these trigrams are used to make query with ANDs and ORs. All the trigrams in the search string are searched against the trigrams inverted index to find the set of candidate documents. After that, a full regular expression search is performed over the set of candidate documents for finding final search results.

According to Cox (2012), converting a search string into a regular expression is not simple. On applying a full set of rules, each rule will compute five sets of results. However, applying all these rules will not produce very meaningful queries. Moreover, these rules will make sets of queries very large and unmanageable. To handle this, simplification rules are applied at each step. After this step, information saving and information discarding steps are applied to keep query sets manageable (Cox, 2012).

2.2 Deliverable-2

The aim of this deliverable is to understand how Naïve Bayes classifier can be used to recognize whether a given search string is from Java or Python. To experiment with Naïve Bayes classifier a program is written in Java.

In the program Java and Python languages are treated as hypothesis. The training set used in the classifier consisted of specific number of Java and Python source code files. All the data from source code files used in the training set was maintained in the two separate text files one for Java one for Python. For simplicity sake these text files are treated like a collection of documents where each document represents content from each respective source code file and each document is separated by '\n\n'. However, '\n' still represents content from same document.

The program splits the contents of Java and Python documents into trigrams and stores them separately. Similarly, search string which is in a form of code snippet is also fragmented into trigrams and stored for future calculation. The program counts the total number to Java and Python documents separately and stores it let say N_{Java} and N_{Python} respectively. The program also individually counts the total number of Java and Python documents containing each of the trigram of training set and stores it in let say $N_{TrigramJava}$ and $N_{TrigramPython}$ respectively. The program does a one-time calculation to find probability of each trigram from the training set. Sample probability calculation is shown in the below table:

Trigrams	Java($N_{TrigramJava}$)	Python($N_{TrigramsPython}$)	$prob_{(TrigramJava)}$	$prob_{(TrigramPython)}$
Trigrams1	$N_{Trigram1Java}$	$N_{Trigram1Python}$	$N_{Trigram1Java} / N_{Java}$	$N_{Trigram1Python} / N_{Python}$
Trigrams2	$N_{Trigram2Java}$	$N_{Trigram2Python}$	$N_{Trigram2Java} / N_{Java}$	$N_{Trigram2Python} / N_{Python}$

The program also calculates the probability for each hypothesis, i.e., Java or Python. Data for calculating probability of hypothesis was hard-coded in the program. These data were obtained by recording total search results obtained on separately typing Java and Python in Google. The program calculates the probably of hypothesis as below:

$$TotalSearchResult = count(JavaSearchResults) + count(PythonSearchResults)$$

$$prob(hypothesis_{Java}) = count(JavaSearchResults) / count(TotalSearchResults)$$

$$prob(hypothesis_{Python}) = count(PythonSearchResults) / count(TotalSearchResults)$$

This model of calculating probabilities of trigrams does not take care of probability of unknown trigram. To resolve this issue, smoothened probability model is used in the program. To calculate probability of unknown trigrams in Java and Python, a random Java and a random Python source code files are used in document representation. The probability calculation for unknown Java and Python trigrams is shown below:

$$prob_{UnknownJava} =$$

$$count(NewTrigrams_{InRandomJavaDocument}) / count(TotalTrigrams_{InRandomJavaDocument})$$

$$prob_{UnknownPython} =$$

$$count(NewTrigrams_{InRandomPythonDocument}) / count(TotalTrigrams_{InRandomPythonDocument})$$

The program then calculates a smoothened probability of each trigram in Java and Python training set by multiplying probabilities of each Java and Python trigrams by $(1 - prob_{UnknownJava})$ and by $(1 - prob_{UnknownPython})$ respectively. The program separately categorizes the trigrams of the query as known and unknown trigram for Java and Python. For each query trigram if it belongs to a set of trigrams from the training set then the program considers it as a known trigram; otherwise, the program considers it as an unknown trigram. This

process is repeated separately for Java and Python. The program finally calculates the probability for the entire query which will decide whether search string belongs to Java or Python. The calculation for the query is shown below; in it α is a constant which is taken as 1, for simplicity sake.

$$prob(QueryTrigrams_{JavaSmooth}) = multiply(SmoothenedJavaProbabilities_{AllKnownJavaTrigrams})$$

$$prob(QueryTrigrams_{PythonSmooth}) = multiply(SmoothenedPythonProbabilities_{AllKnownPythonTrigrams})$$

$$prob(trigrams_{JavaUnknown}) = multiply(UnknownJavaProbabilities_{AllUnknownJavaTrigrams})$$

$$prob(trigrams_{PythonUnknown}) = multiply(UnknownPythonProbabilities_{AllUnknownPythonTrigrams})$$

$$prob(classifier_{Java/Query}) =$$

$$\alpha \times prob(QueryTrigrams_{JavaSmooth}) \times prob(Trigrams_{JavaUnknown}) \times prob(hypothesis_{Java})$$

$$prob(classifier_{Python/Query}) =$$

$$\alpha \times prob(QueryTrigrams_{PythonSmooth}) \times prob(Trigrams_{PythonUnknown}) \times prob(hypothesis_{Python})$$

If $prob(classifier_{Java/Query}) > prob(classifier_{Python/Query})$ then the given query belongs to Java programming language.

Else if $prob(classifier_{Python/Query}) > prob(classifier_{Java/Query})$ then the given query belongs to Python programming language.

Results from the Program are shown below:

```
Probability (query is from Java)------->0.0000014471691947643424669702356827103253616716069093701460943648544682438867246123264818617525574912470 46
Probability (query is from Python)----->1.369806853992798135544873114648216951387206612833414855085594834937974465337790469540447213892971644E-8
Result----->Query entered belongs to Java programming language
```

Figure1: Results with five source code documents in Java and Python in the training set

```
Probability (query is from Java)------->0.000073948021774385616364157191000850821735747582196687542533408750518175400149610327658636320325324178 50
Probability (query is from Python)----->7.160896577656772668251697512087502702036814721534614888924208334252895558880542932624579939077937864E-16
Result----->Query entered belongs to Java programming language
```

Figure2: Results with ten source code documents in Java and Python in the training set

The conclusion that I draw from this experiment is that result produced by a Naïve Bayes classifier improve as size of training set increases.

2.3 Deliverable-3

This deliverable has two parts. The goal of the first part of the deliverable is to learn PHP and the goal of second part of the deliverable is extend the capability of Yioop's text processor to process Java and Python source code files.

For the first part of the deliverable, the Naïve Bayes classifier of deliverable-2 was recoded in PHP. This helped me to learn the basic concepts of PHP. This assignment revealed the basic difference between various components available in Java and PHP. One of the important structures in PHP is an array; an array in PHP allows values in the form of key value pair whereas value in an array can be another array. Arrays in PHP were used to store the trigrams and various values associated with the trigrams like individual trigram probabilities, smoothened trigram probabilities and document count for trigrams.

To handle large floating point numbers in the program, I used the BCMath library which is available in PHP. BCMath library supports many algebraic operations with large floating point numbers. It also helps in calculating various values PHP in coded Naïve Bayes classifier. The final output of the PHP coded Naïve Bayes classifier was in-line with the Java coded Naïve Bayes classifier for same set of Java and Python source code files in the training set. Aptana Studio 3.0 was used as IDE to develop the PHP program for classifier.

Results from the Program are shown below:

```
sridhar-chunduris-MacBook-Pro:GIT sridharchunduri$ php classifier.php java.txt javaRandom.txt python.txt pythonRandom.txt "System.out.println"
Probability (query is from Java)------->0.000073948021774385616364157191000850821735747582196687542533408750518175400149610327658636320325324 2
Probability (query is from Python)----->0.000000000000000716089657765677266825169751208750270203681472153461488892420833425289555888054293262 5
Result----->Query entered belongs to Java programming language
```

Figure 3: Results with ten source code documents in Java and Python in the training set

For the second part of the deliverable, I have created locale tags in Yioop for Java and Python. In tokenizer.php of Java and Python chargram value is set to three to chunk the crawled Java and Python source code files into trigrams. I have also modified Yioop 's config.php to call

Text Processor when the mime types of Java and Python source code files are discovered. This change will make Yioop's internal function to call Text Processor when Java or Python source code files are crawled.

When source code files are crawled then their extensions are extracted in the calculateLang method of the TextProcessor.php. Based upon the extension, the language is set for the crawled file. When Yioop's internal operations find the $lang variable is Java then the crawled Java source code file will be chunked into trigrams. Similarly, when Yioop's internal operations find that the $lang variable is Python then the crawled Python source code file will also be chunked into trigrams.

2.4 Deliverable-4

The main purpose of this deliverable is to create a PHP program which reproduces the effect of Git clone with the help of cURL calls. Git clone is a Git command used for copying the files from remote repository into the local machine.

The first step to accomplish the goal of the deliverable was to find out the requests made by the local machine to the remote repository. To find this out a local Git repository was configured with help of WebDav. For simplicity sake, I have used XAMPP.

To setup the local Git repository, first of all I have installed XAMPP, after this I have created repository.git directory inside the XAMPP's document root. It is very important to give write permissions to the new local repository created and to the xamppfiles folder inside XAMPP. After providing the write permissions, I have modified the httpd.conf file which is present inside etc folder of XAMPP to uncomment WebDav and to enable dav for the local Git repository. I have made this new local Git repository as bare repository and updated the server information. The directory structure of a local Git repository is shown below:

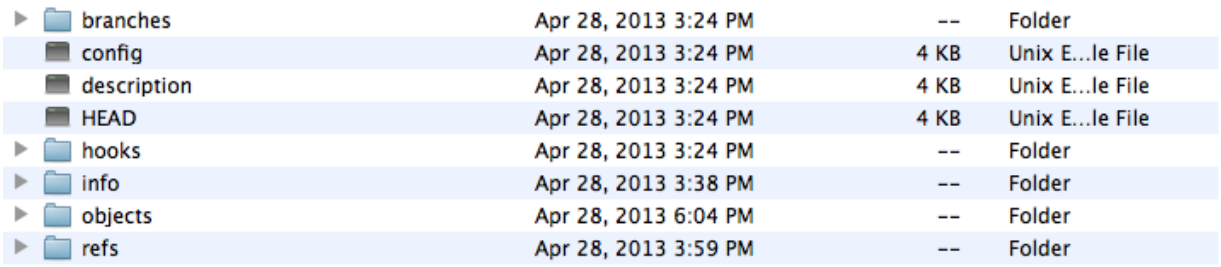

Figure 4: Local Git repository structure in Mac OS X

To test my PHP-based Git clone program, I have arranged some sample Java and Python files inside a nested folder structure. Next, I added and committed these files using Git commands and pushed these committed file to a local Git repository by using Git push

command. Finally, I cloned the files which I pushed previously to trace the requests made while cloning the files from the repository. XAMPP access logs give the list of requests made.

The screenshot below shows sample GET requests used for cloning the files from local Git repository.

```
127.0.0.1 - - [13/May/2013:03:12:30 -0700] "GET /repository.git/info/refs?service=git-upload-pack HTTP/1.1" 200 59
127.0.0.1 - - [13/May/2013:03:12:30 -0700] "GET /repository.git/HEAD HTTP/1.1" 200 23
127.0.0.1 - - [13/May/2013:03:12:30 -0700] "GET /repository.git/objects/0d/b099e7c42e2df89d5ba3ecdb9a462698596eb5 HTTP/1.1" 200 151
127.0.0.1 - - [13/May/2013:03:12:30 -0700] "GET /repository.git/objects/f1/d6123f6e704e4c5ab6e26dbca39516f53aba34 HTTP/1.1" 200 774
127.0.0.1 - - [13/May/2013:03:12:30 -0700] "GET /repository.git/objects/03/755be2b05f2cf9ec159992ad362a01283b313e HTTP/1.1" 200 9186
127.0.0.1 - - [13/May/2013:03:12:30 -0700] "GET /repository.git/objects/41/79fa2b3cb7d34039a2a47669c0bd643e5ad7c1 HTTP/1.1" 200 154
```

Figure 5: Sample GET requests for Git clone

The next step towards achieving the goal would be to make these GET requests using cURL via a PHP program. The example below indicates the steps used by the PHP program to reproduce the effects of Git clone operation.

My PHP program makes the first cURL request to obtain the contents from the first GET request of the Figure 5. Output of the cURL request is shown below:

```
sridhar-chunduris-MacBook-Pro:GIT sridharchunduri$ php GitClone.php
0db099e7c42e2df89d5ba3ecdb9a462698596eb5        refs/heads/master
```

Figure 6: Output obtained from cURL call to the first GET request of the Git clone

After analyzing the output of cURL request as shown in Figure 6, I found that he output of first GET request of the Git clone operation indicates the SHA hash of the Git object in hexadecimal. Third row of the Figure 5 contains this hexadecimal SHA hash. The cURL call to the second GET request of the Figure 5 simply confirms the branch as master. After exploring how Git stores the object, I found that Git stores data in two kinds of objects namely blob and tree object. The blob object of Git contains the actual data in zlib compressed format. The tree object of Git contains structural information in compressed format. Uncompressed output of the cURL call to the third GET request of the Figure 5 is shown in the next page:

```
sridhar-chunduris-MacBook-Pro:GIT sridharchunduri$ php GitClone.php
commit 228tree f1d6123f6e704e4c5ab6e26dbca39516f53aba34
author Snigdha Parvatneni <snigdha.parvatneni@gmail.com> 1368439403 -0700
committer Snigdha Parvatneni <snigdha.parvatneni@gmail.com> 1368439403 -0700

setting up initial repository
```

Figure 7: Uncompressed output of cURL call to the third GET request of the Git clone

Again analyzing the output of the cURL request as shown in Figure 7, I found that it has the SHA hash of the Git object as indicated by the fourth row of the Figure 5. Uncompressed output obtained from the cURL call to the fourth GET request of Figure 5 indicates presence of binaries in it. It represents the Git tree object. Hex dump of the output indicated a useful pattern. Screenshot of the portion of the hex dump of the output obtained from the cURL call to the fourth GET request of Figure 5 is shown below:

```
sridhar-chunduris-MacBook-Pro:~ sridharchunduri$ hexdump -C /Users/sridharchunduri/Desktop/hex.txt
00000000  74 72 65 65 20 31 30 32  39 00 31 30 30 36 34 34  |tree 1029.100644|
00000010  20 41 63 74 69 6f 6e 42  61 72 2e 6a 61 76 61 00  | ActionBar.java.|
00000020  03 75 5b e2 b0 5f 2c f9  ec 15 99 92 ad 36 2a 01  |.u[.._,......6*.|
00000030  28 3b 31 3e 34 30 30 30  30 20 4a 61 76 61 00 41  |(;1>40000 Java.A|
00000040  79 fa 2b 3c b7 d3 40 39  a2 a4 76 69 c0 bd 64 3e  |y.+<..@9..vi..d>|
00000050  5a d7 c1 34 30 30 30 30  20 50 79 74 68 6f 6e 00  |Z..40000 Python.|
00000060  75 b0 a0 18 22 09 03 9f  f3 60 05 34 5b 88 9b 86  |u..."....`.4[...|
00000070  71 3c 06 de 31 30 30 36  34 34 20 53 68 65 72 6c  |q<..100644 Sherl|
00000080  6f 63 6b 41 63 74 69 76  69 74 79 2e 6a 61 76 61  |ockActivity.java|
00000090  00 7b 45 43 64 05 1d bb  5c a6 3d e8 49 8c 1b 29  |.{ECd...\.=.I..)|
000000a0  37 c3 35 c2 a4 31 30 30  36 34 34 20 53 68 65 72  |7.5..100644 Sher|
000000b0  6c 6f 63 6b 44 69 61 6c  6f 67 46 72 61 67 6d 65  |lockDialogFragme|
000000c0  6e 74 2e 6a 61 76 61 00  a7 c8 56 bf 02 f9 a2 c6  |nt.java...V.....|
```

Figure 8: Portion of the hex dump of the Git tree object

According to Schwarz (2010) "the Git tree contains the internal representation of Git's directory structure. The general format of Git tree object is represented by: tree ZN(A FNS)*. Here Z represents the size of the objects in byte, N represents the null character, A indicates the UNIX access code, F represents the file name, and S indicates 20 bytes long SHA hash. The same pattern repeats for each Git object." I have observed UNIX access code of 1600644 and 100755 for the Git blob objects and UNIX access code of 40000 for Git tree objects. In the local Git repository Git objects are stored inside the objects folder where each object has a folder named with first two bytes of SHA hash and the remaining

38 bytes shows the file name. The snap shot indicating the structure of the object folder in local Git repository is shown below:

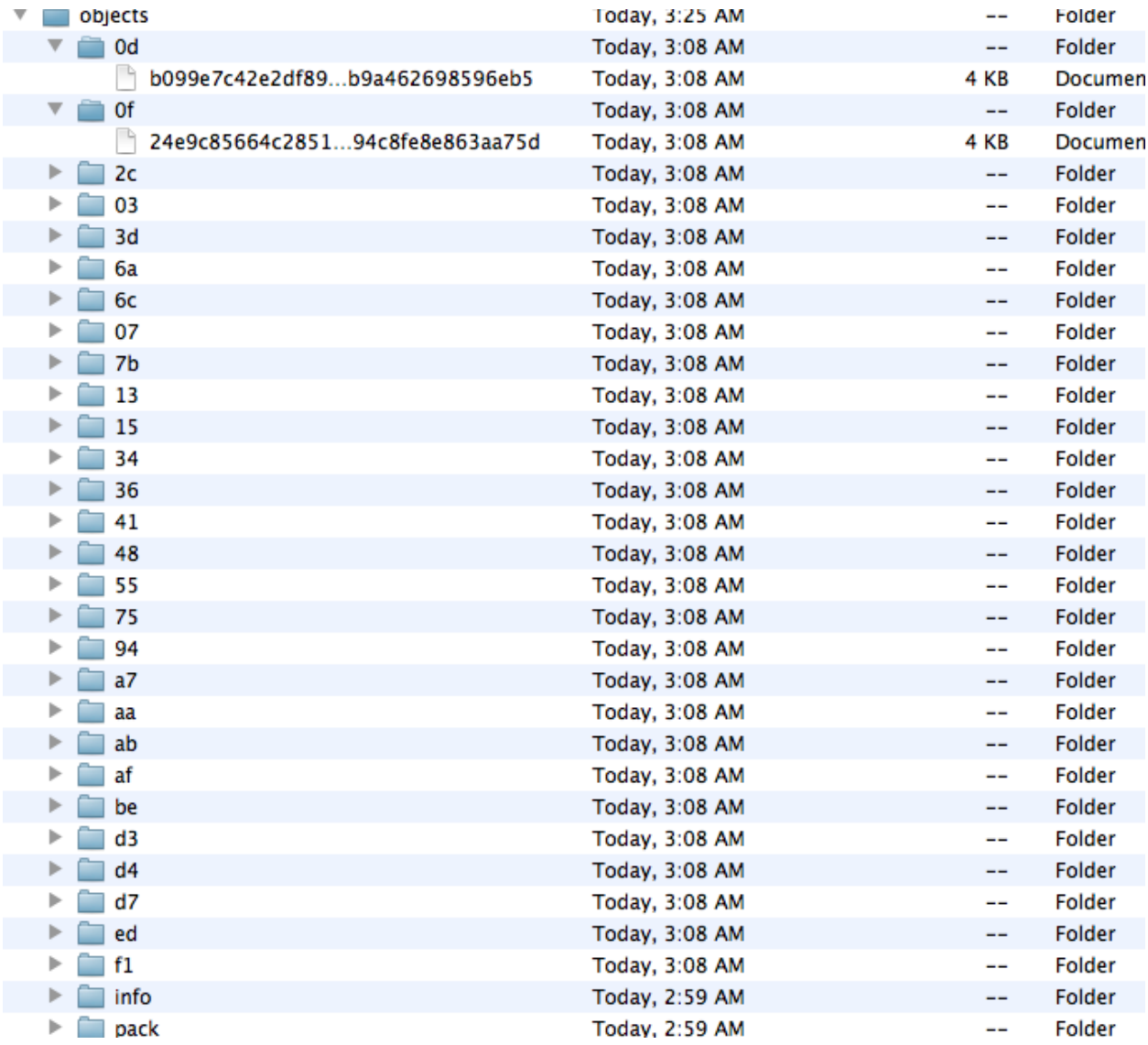

Figure 9: Object folder structure for local Git repository in Mac OS X

With help of the knowledge of Git tree objects, the program extracts the SHA hash of each Git object and makes the cURL call to get the actual content of each Git object. Each Git tree object contains the roadmap for the files and folders included inside it; whereas, details of each folder will be inside its own tree object and the actual contents of the files will be inside Git blob objects. The program makes successive cURL requests for getting the compressed contents

of all the Git blob objects. After getting the compressed contents of Git blob files, the program decompresses the compressed contents. Then the program determines the nested folder structure with help of the information contained inside Git tree objects. The Git tree object contains different access codes for files and folders. The Git tree object also contains names for files and folders. Finally, the program creates the files inside the nested folder structure and finally writes the uncompressed contents to their respective files.

3. CONCLUSION

The deliverables in my CS297 project helped me to understand the different concepts that will be needed in CS298. Deliverable-4 provided me with the hands on experience of downloading source code from a Git repository, which will be used in the CS298 project to download open source code files from remote Git repositories. Search operations will be performed over an inverted index built from these downloaded source code files. Deliverable-2 gave me an opportunity to understand and implement the Naïve Bayes classifier, which will be used in CS298 to recognize the language of a search query. Deliverable -3 helped me find a way to set the language for crawled pages and use this language settings to chunk the contents of the page into trigrams. The entire project of implementing the source code searching feature in Yioop will be completed in Fall 2013 for the CS298.

REFERENCES

Bajracharya, S., Baldi, P., Dou, Y., Linstead, E., Lopes, C. Ngo, T., & Rigor, P. (2006, October). *Sourcerer: A search engine for open source code*. Paper presented at Proceedings of the 2006 Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Oregon, Portland, USA. Retrieved on February 12, 2013 from http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&sqi=2&ved=0CEMQFjAC&url=http%3A%2F%2Fwww.researchgate.net%2Fpublication%2F228804354_Sourcerer_A_Search_Engine_for_Open_Source_Code%2Ffile%2F79e4150930483860a6.pdf&ei=VWaQUfSMEeOQjAKgwIGAAg&usg=AFQjCNHLSP_h9lrEkiJysTyVdZhLlCczgg&bvm=bv.46340616,d.cGE

Cox, R. (2012, January). *Regular Expression Matching with a Trigram Index*. Retrieved on February 15, 2013, from http://swtch.com/~rsc/regexp/regexp4.html

Schwarz, N. (2010, March 25). *Niko Schwarz's science and programming: Git tree objects, how are they stored?* Retrieved April 20, 2013, from http://smalltalkthoughts.blogspot.com/2010/03/git-tree-objects-how-are-they-stored.html