Adding Source Code Searching Capability to Yioop

Advisor - Dr Chris Pollett

Committee Members – Dr Sami Khuri and Dr Teng Moh

Presented by

Snigdha Rao Parvatneni

AGENDA

- Introduction
- Preliminary work
- Git Clone effects in Yioop
- Source Code Searching Techniques
  - Logarithmic Char-gramming
  - Suffix tree
- Comparing both the techniques in Yioop
- Conclusion

INTRODUCTION

- Code search enables users to search open source code.
- Code snippets can be used as a query string.
- Source code search helps users in finding specific implementations over large collection source code in open source repositories.
- Some examples of available code search engines are Ohloh, Google code, Krugle etc.
- This project aims to implement Java and Python source code search in Yioop, using publically crawlable Git repositories.

TECHNIQUES FOR CODE SEARCH

- Two approaches of code search experimented in Yioop are:
  - Logarithmic Char-Gramming
  - Suffix Tree
- The logarithmic char-gramming technique was new to Yioop. A native approach of calculating character n-grams is available in Yioop.
- A suffix tree implementation was already present in Yioop and was extended for source code search.
- Famous Git hosting web servers like GitHub, Gitorious, etc., are not publically crawlable and hence cannot be ethically used.

PRELIMINARY WORK

- Individual components of code search were separately implemented to get an overall idea about an actual implementation of the feature in Yioop.
- Proof of concept was developed for
  - Naïve Bayes classifier – to programmatically detect the language of a query string.
  - Git cloning effect – to clone a Git repository without using the Git clone command or any other external utilities
- The proof of concepts were created using PHP and experiments were conducted to better understand the concepts.

NAÏVE BAYES CLASSIFIER

- A Naïve Bayes classifier was implemented to detect the language of a query string.
- In the classifier, Java and Python programming languages are treated as hypotheses.
- The classifier's training set consists of Java and Python source code in a document representation, where each document is separated by '\n\n'.
- Source code were chunked into trigrams and the initial probabilities of hypotheses were calculated.

NAÏVE BAYES CLASSIFIER CONTD…

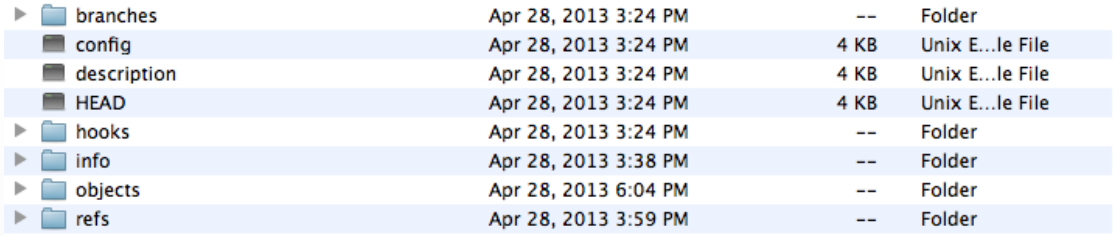
- To calculate the probability of an unknown trigram random Java and Python documents were taken.
- The probability of unknown trigrams were calculated by dividing the number of new trigrams in a random document by the total number of trigrams in a random document.
- Probabilities of trigrams are smooth by multiplying the initial probabilities of trigrams by one minus the probability of an unknown trigram.

NAÏVE BAYES CLASSIFIER CONTD…

- The probability of hypothesis is calculated by dividing the total number of search results of each hypothesis by the total number of search results of both the hypotheses.
- A query string is chunked into trigrams.
- The final probability of a query is obtained by multiplying the probabilities of known and unknown query trigrams with the probability of hypotheses.
- The larger probability value decides the language of a query.

GIT REPOSITORY STRUCTURE

- Git is a popular open source version control system.
- The Git clone is a Git command for copying files from a remote repository.
- The Git clone command was reverse engineered to download source code.
- To experiment a local Git repository was configured with help of WebDav and source code were pushed.



A local Git repository structure in Mac OSX

INTERNAL REPRESENTATION OF GIT DIRECTORY STRUCTURE

- The general format of a Git tree object is represented by: tree ZN(A FNS)*

  Z represents the size of the objects in byte

  N indicates the null character

  A denotes the UNIX access code

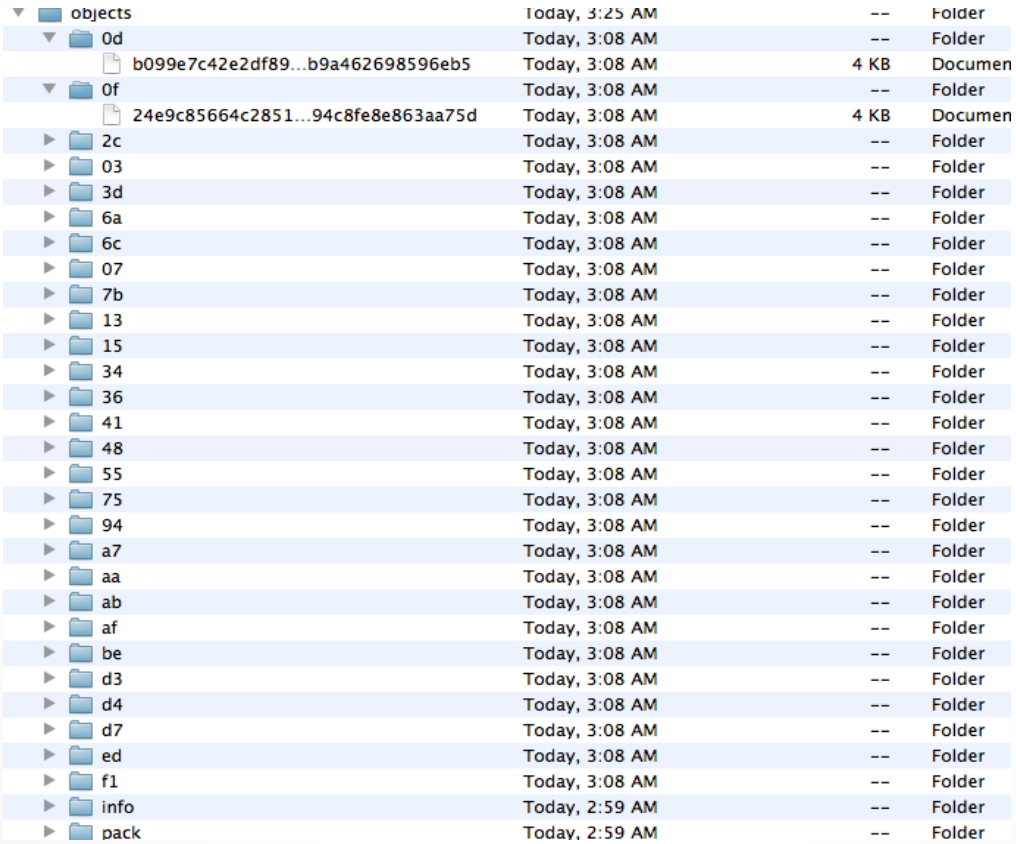
  F represents the file name

  S indicates 20 bytes long SHA hash

- The first two bytes of SHA hash represent the folder name and the remaining 38 bytes indicate the file name.

GIT OBJECT FOLDER STRUCTURE

- Objects folder contains the actual Git blob and tree objects.



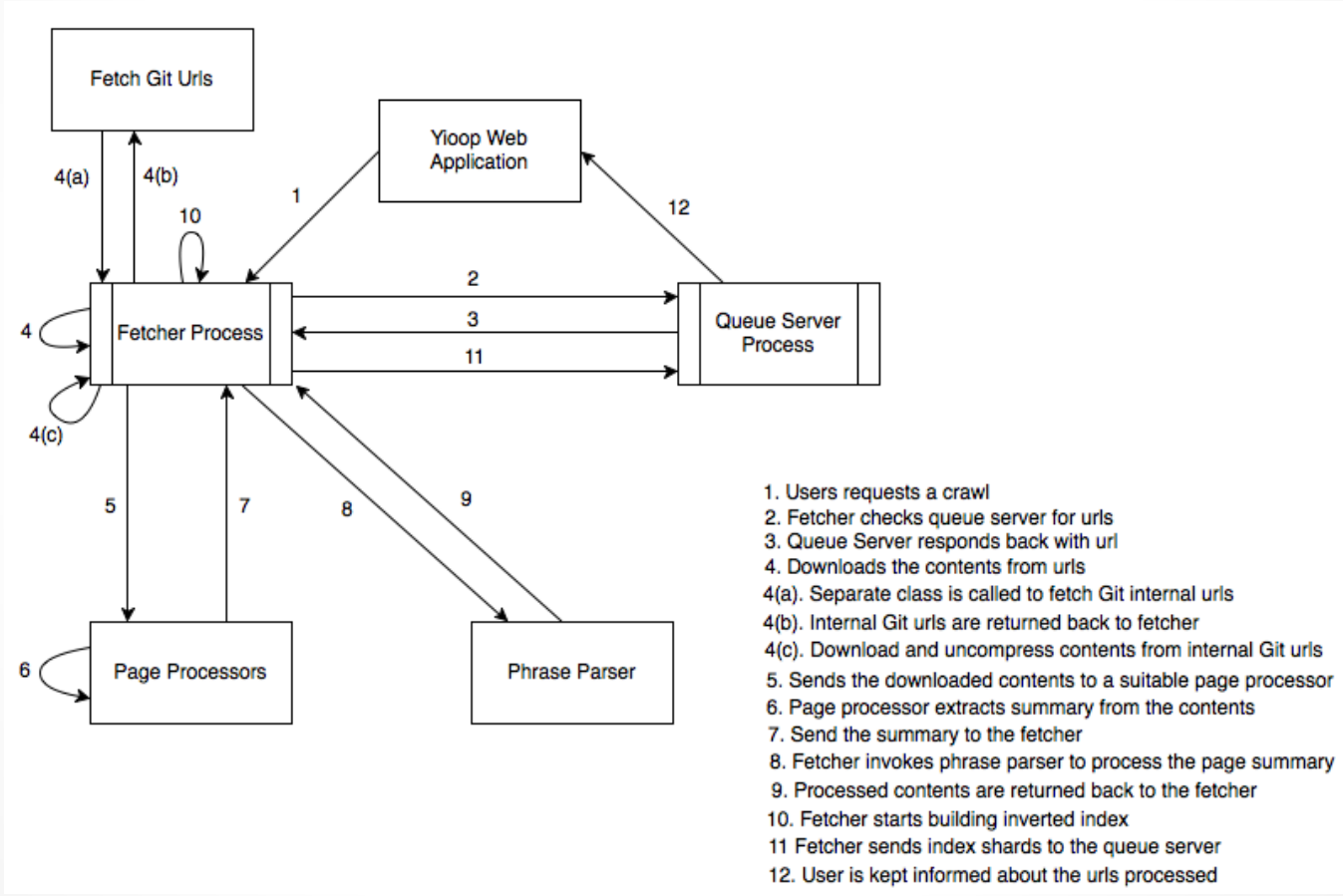
GIT CLONE USING cURL REQUESTS

- cURL request to each Git internal url provides the next Git url.
- The first Git url can be formed by appending the Git url with a fixed component “info/refs?service=git-upload-pack”
- Git Blob objects contain the actual content of the file in a compressed manner.
- Git tree objects contain the information about the organization of Git blob objects.
- A cURL requests was made to get the compressed content from a Git object. The content received was uncompressed to get the actual content.

GIT CLONNING EFFECTS IN YIOOP

- In Yioop, a fetcher process fetches the urls and downloads contents from each url.
- These downloaded contents are processed based on their type. The fetcher then builds an inverted index using these processed contents.
- When Yioop encounters a Git url, then the Git internal urls are fetched from the parent Git url and contents are downloaded from these urls and uncompressed.
- After all the Git urls are downloaded the control returns back to the normal routines of fetching urls

GIT CLONE IMPLEMENTATION IN YIOOP



1. Users requests a crawl
2. Fetcher checks queue server for urls
3. Queue Server responds back with url
4. Downloads the contents from urls

4(a). Separate class is called to fetch Git internal urls

4(b). Internal Git urls are returned back to fetcher

4(c). Download and uncompress contents from internal Git urls

5. Sends the downloaded contents to a suitable page processor
6. Page processor extracts summary from the contents
7. Send the summary to the fetcher
8. Fetcher invokes phrase parser to process the page summary
9. Processed contents are returned back to the fetcher
10. Fetcher starts building inverted index
11 Fetcher sends index shards to the queue server
12. User is kept informed about the urls processed

LOGARITHMIC CHAR-GRAMMING

- Logarithmic char-gramming is a modification of a char-gramming technique.
- A char-gramming technique is used to process text that contains a contiguous sequence of characters.
- Character n-grams are the chunks of continuous text each of size n. For example, if the text is “shining bell” and , n = 3 then 3-grams extracted from the text are “shi ,hin, ini, nin, ing, ng_, g_b, _be, bel, ell”
- In the logarithmic char-gramming, a text is chunked into character n-grams where n starts from 3 and keeps doubling until it exceed the length of the text.

LOGARITHMIC CHAR-GRAMMING

- For the text “shining bell”, the value of n starts from 3 and doubles to 6 and then doubles to 12. Here, the length of the text is 12 therefore, doubling stops when the n reaches 12.
- The character n-grams produced for the text “shining bell” in the logarithmic char-gramming technique are:

  3-grams - “shi ,hin, ini, nin, ing, ng_, g_b, _be, bel, ell”

  6-grams - “shinin, hining, ining_, ning_b, ing_be, ng_bel, g_bell”

  12-grams - “shining_bell”

SUFFIX TREE

- A suffix tree is a tree-based data structure which contains all the suffixes of a given string.
- Yioop has an implementation for the Ukkonen's algorithm to build a suffix tree.
- In Yioop, the newly introduced source code tokenization processes provide terms needed to build a suffix trees for source code.
- Each term from the source code act as an alphabet while building the suffix tree.

TOKENIZING JAVA AND PYTHON SOURCE CODES

- Java and Python source code have definite structures and organization of words. These characteristics of Java and Python source codes can be used to tokenize the source code into lexical units.
- The lexical structure of the Java and Python programming languages are different.
- In this approach the focus is to split the source code into tokens and to build suffix trees using these tokens.
- Earlier, in Yioop there was no specific implementation to construct suffix tree for source code.

JAVA TOKENS

- Token in Java can be categorized into
  - Keywords
  - Identifiers
  - Separators
  - Operators
  - Comments
  - Literals
- Literal is again categorized into integer literal, floating-point literal, character literal, string literal, boolean literal and null literal.

PYTHON TOKENS

- Token in Python can be categorized into
  - Identifiers
  - Keywords
  - Operators
  - Delimiters
  - Comments
  - Literals
- Literal is again categorized into numeric literal, floating-point literal, logical literal, string literal, byte literal and none type literal.

MAXIMAL AND CONDITIONALLY MAXIMAL SUB-STRINGS

- For each source code file, Yioop builds a suffix tree from tokenized source code and then finds the maximal and conditionally maximal sub-strings.
- A string is called a maximal string if it does not act as prefix of any other string in the document and all the occurrences of a given string includes other strings in the document.
- A string is called a conditionally maximal string if it acts as a prefix of maximal string in a document and there is no other string in the document which lies between them.
- Yioop, stores all the maximal sub-strings along with the pointers to their respective conditionally maximal strings.

EXAMPLE

Document d_1: 12341235													
Maximal Sub-Strings	1	2	3	4	5	123	12341235	23	2341235	341235	41235	1235	235
Conditionally Maximal Sub-Strings	/	/	/	/	/	1	123	2	23	3	4	1	2

Document d_2: 123456											
Maximal Sub-Strings	1	2	3	4	5	6	123456	23456	3456	456	56
Conditionally Maximal Sub-Strings	/	/	/	/	/	/	1	2	3	4	6

EXAMPLE

- In the tables “/” indicated the root element.
- For query q_1= 12 which never appears as a maximal string for any of the above documents.
- Yioop looks for the cases where 1 occurs as a conditionally maximal sub-string and is followed by 2.
- The documents, which satisfy this condition, are returned as the search results.

SOURCE CODE QUERYING METHODS

- In Yioop, an inverted index is used to perform search operations.
- Naïve Bayes classifier is a probabilistic model, so at times it detects language incorrectly.
- Incorrect language detection for search strings affects Yioop’s performance negatively in terms of returning relevant results.
- To avoid this we have used control words.
- The control words are used to explicitly mention the language of the code snippet or query string.

QUERYING METHOD FOR LOGARITHMIC CHAR-GRAMMING

- The querying technique in the logarithmic char-gramming approach finds two largest char-grams from the query string.

$$length_1 = length(query)$$

$$k_1 = \left\lfloor \log \left\lfloor \frac{length_1}{3} \right\rfloor \right\rfloor$$

$$X = 3 \times 2^{k_1}$$

where X indicates length of the segment of the query string from the beginning

$$length_2 = length - X$$

$$k_2 = \left\lfloor \log \left\lceil \frac{length_2}{3} \right\rceil \right\rfloor$$

$$Y = 3 \times 2^{k_2}$$

where Y indicates length of the segment of the query string from the end

QUERYING METHOD FOR LOGARITHMIC CHAR-GRAMMING

- Two largest char-grams from a given query string is found by splitting the original query string into two sub-strings; one from beginning of the string for length X and another from ending of the string for length Y.
- These two char-grams are searched over an inverted index to find a match.
- Source code files for which match is found are returned to a user as relevant search results.

QUERYING METHOD FOR SUFFIX TREE

- In the suffix tree approach, the querying technique is exactly same as the indexing technique.
- A query string is tokenized into tokens to build a suffix trees.
- Maximal and conditionally maximal substrings were calculated to return the matching results.

COMPARING PERFORMANCE IN YIOOP

- The crawl performance statistics for the logarithmic char-gramming approach of code search

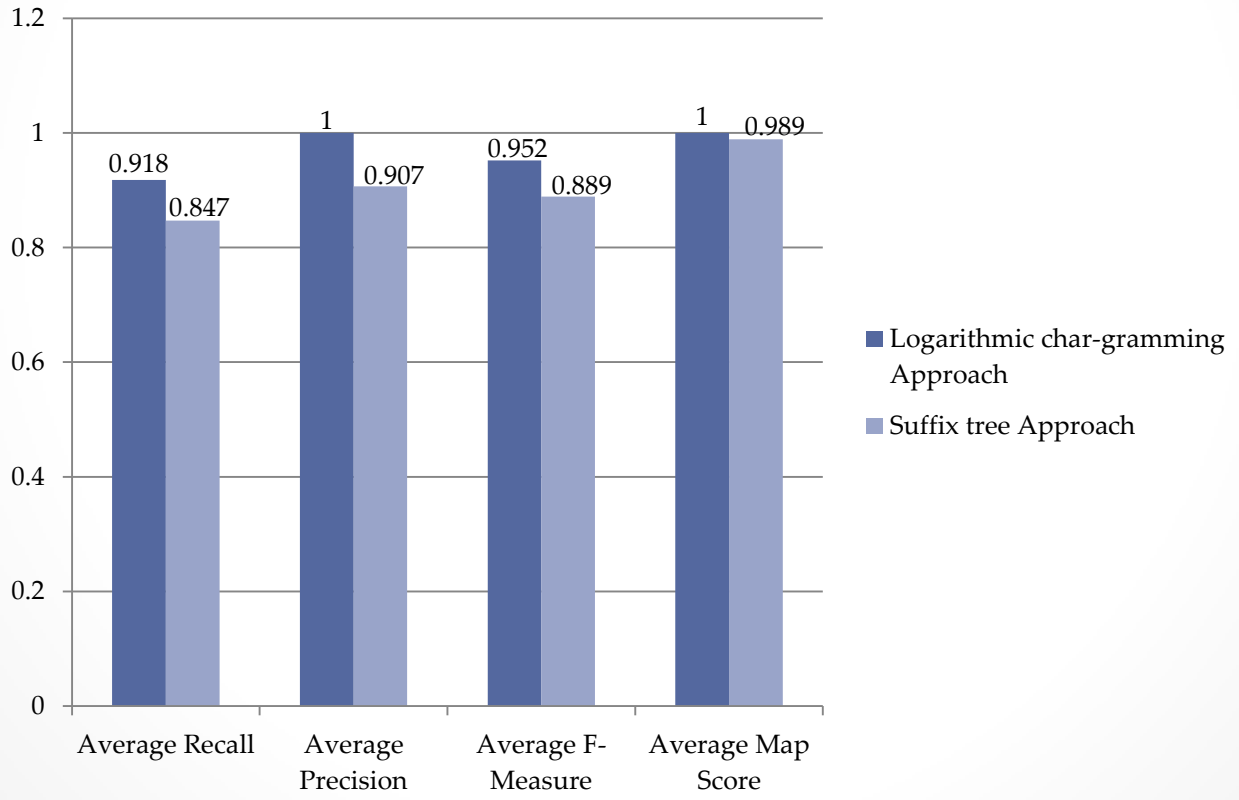
Number of files in Git repository	Size of the inverted index	Time taken to build inverted index in HH:MM:SS	Memory usage	Original size of files
10	81.2 MB	00:06:55	1158373144	193KB
100	359.1 MB	00:25:50	1742086984	1.1MB
1000	1.75 GB	01:42:47	1868283592	8.2MB

- The crawl performance statistics for the suffix tree approach of code search

Number of files in Git repository	Size of the inverted index	Time taken to build inverted index in HH:MM:SS	Memory usage	Original size of files
10	3.2 MB	00:00:51	515064424	193KB
100	11.1 MB	00:02:24	515065288	1.1MB
1000	57.6 MB	00:08:15	686648216	8.2MB

COMPARING EFFECTIVENESS IN YIOOP

- Average values for different effectiveness measures



SOURCE CODE SEARCHING TECHNIQUES IN YIOOP

- On the basis of both performance and effectiveness of code search techniques implemented in Yioop, we have decided that suffix tree technique is a reasonable approach to search source code in Yioop.
- In terms of effectiveness, the logarithmic char-gramming technique performs a little bit better than the suffix tree technique. However, in terms of performance suffix tree technique performs much better than logarithmic char-gramming technique.
- Therefore, suffix tree approach is selected as a result of trade-off between performance and effectiveness.

CONCLUSION

- In this project a Java and Python source code search feature was implemented in Yioop.
- Our technique of code search addresses the complexity of source code search without deteriorating the performance of Yioop.
- It was decided to add the suffix tree implementation of code search to the main Yioop branch.
- Yioop could easily support the addition of other programming languages.
- All an implementer needs to do is code a programming language specific tokenizer and page processor in Yioop.

THANK YOU