# BUILDING A MINI SEARCH ENGINE

## Indexing Phase

**By**

**Rajendra Banjade**

**The University of Memphis, TN, USA**

**2012**

A Perl program that generates an inverted index of a set of already preprocessed files. The files are stored in a directory which is given as an input parameter to the program. Use the programs described in previous steps and you will have preprocessed texts ready to fed into this program.

```
# Creates inverted index from the set of preprocessed files.
#   - For each term in vocabulary, creates a list of files that term appears
along with the frequency.
#   - calculates df (document frequency), idf (inverted document frequency -
log2 (N/n))
#   - calculates the document length.
#
# VERSION HISTORY:
# Rajendra Banjade 10/15/2012
# The University of Memphis, TN, USA
#

use Data::Dumper;                       # for formatted printing of various
datastructures including Hash.

my $docCount = 0;                       # counter for processed files.
my $tokenCount = 0;                     # total number of index terms in the
corpus.

# main inverted index structure
%indexHash = ();
# hash for storing document vector length.
%docLengthHash = ();
# document frequency hash <token, df>
%dfHash = ();
# idf hash <token, idf>
%idfHash = ();

# subroutine to calculate log base 2. log function gives the natural log and
# we apply basic algebra to calculate log base 2.
sub log2($) {
    my $n = shift;
    return log($n)/log(2);
}

# For each preprocessed file, iterates over the terms and creates inverted
index.
opendir(PPDIR, "../preprocessed") || die "Can't opedir: $!\n";
while (readdir(PPDIR)) {
    $file = ($_);
    next unless (!($file =~ /^\./));              # skip file that starts
with ., some hidden files.
    open (PPFILE, "<../preprocessed/$file") || die ("Failed to ope
:preprocessed/$file");

    $fileContent = join (" ", <PPFILE>);        # bring whole content in a
```

```
single string.

    my @tokens = split /\s+/, $fileContent;     # split by whitespace(s)
    $tokenCount += @tokens;                     # count the tokens.

    $file =~ s/\.txt//g;                        # just put the document name
as id (removing file extension).

    # foreach token, create/update inverted index.
    foreach my $token (@tokens) {
        my %docHash = ();
        if (exists $indexHash{$token}){
            %docHash = %{$indexHash{$token}};
        }
        # update the term frequency. If hash doesn't contain $file,
        # it adds and initializes to zero (no need to check explicitly).
        $docHash{$file}++;

        # update the index hash as document and/or term frequency hash has
been updated.
        $indexHash{$token} = \%docHash;
    }
    # close the file handler
    close (PPFILE);
    $docCount++;
}
closedir(PPDIR);

# calculate the tf, idf
foreach my $token (keys %indexHash) {
    my $df =  keys %{$indexHash{$token}};
    $dfHash{$token} = $df;
    #idf is log2(N/n), N - total number of documents and n - document
frequency.
    $idfHash{$token} = &log2($docCount/$df);
}

# calculate document vector length.
foreach my $token (keys %indexHash) {
    $idf = $idfHash{$token};
    my %docHash = %{$indexHash{$token}};
    # sum of square of weight of each term in the document vector.
    # need to calculate the square root of the result (please see the next
loop).
    foreach my $docId (keys %docHash) {
        $docLengthHash{$docId} += ($docHash{$docId}*$idf)**2;
    }
}
# take the square root of sum of squares of term weights (calculated above).
foreach my $docId (keys %docLengthHash){
    $docLengthHash{$docId} = sqrt($docLengthHash{$docId});
}

# Print the values. We can use Dumper for formatted output but here we want
to mix up from different
# hashes for readability and conciseness.
# prints '<token>' df idf { list of <document-id ==> tf> }
```

```perl
foreach my $token (keys %indexHash) {
    my $df = $dfHash{$token};
    my $idf = $idfHash{$token};
    print "'$token' $df $idf\n      {\n";
    foreach my $doc (keys %{$indexHash{$token}}) {
        print "          $doc => " . $indexHash{$token}->{$doc} . "\n";
    }
    print "\n       }\n";
}

# print the formatted output using Dumper.
print "\n ======= Document length ============ \n";
print Dumper (\%docLengthHash);

# print some statistics
print ("\n\n Total files processed: $docCount , Total token count:
$tokenCount, Vocabulary size: ".keys(%indexHash)."\n\n");
```

src/indexer.pl

**NOTE**:

- Perl provides a package Data::Dumper to print the content of various data structures including hash, hash of hash etc. However, dumper has been used only once in this program to print the document vector length. Printing inverted index has been done without using it. Please install this package using perl module manager if that is not available (however, it usually does).
    - Go to CPAN client (in windows 7, please run it as administrator).
    - Issue a command, install Data::Dumper

To run the program,

- Go to the <your folder>/src
- Issue command, perl indexer.pl > dump.txt
  (Redirect console output to a file for readability).

The program,

1. Reads each preprocessed file from the folder../*preprocessed,* creates inverted index of tokens.
2. Calculates document frequency and inverted document frequency.
3. Calculates document vector length for each preprocessed file (though this is not explicitly mentioned in the assignment. It is depends on the ranking algorithm).

**Sample output:**

For inverted index,

*<index word> <df > <idf> {*

*List of <document id> => <tf>*

*}*

```
'thinker' 1 4.32192809488736
        {
            d10 => 1

        }
'lacklust' 2 3.32192809488736
        {
            y8 => 1
            y4 => 1

        }
'aid' 1 4.32192809488736
        {
            y1 => 3

        }
'email' 4 2.32192809488736
        {
            d7 => 1
            d4 => 1
            d6 => 1
            d8 => 1

        }
…
..
```

For document length, *<Document id> => <document vector length>*

```
          'y1' => '101.88973299491',
          'd2' => '6.73791886999699',
          'y8' => '129.509009587872',
          'y6' => '161.96960717419',
          'd8' => '172.103014288449',
          'y9' => '111.008854469144',
          'y4' => '40.6397092151676',
          'y3' => '78.7020049688313',
          'd10' => '626.078661782212',
….

Total files processed: 20 , Total token count: 10523, Vocabulary size: 2967
```