Perl Based Ms-word Documents Search Engine

Based on **Full Text Searching in Perl** by Tim Kientzle

Dr.Dobbs Jan 1999

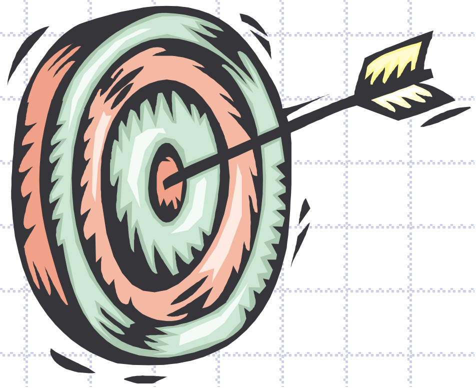
Roey Almog – Perl Mongers meeting on 12 June 2003
roey@arnet.co.il

Agenda

- What was the mission?
- The solution.
- Alternatives & why was Perl used?
- What are the main components?
- Steps of implementation (long so we'll probably skip some parts).
- Summary.

The Mission

- A solution to perform keyword search in a collection of about ***30,000*** MS-Word documents.
- The files are spread ***all over*** the network.
- The first solution was MS Windows built in search that was slowwwwww. And does not support keyword search (as far as I know…).

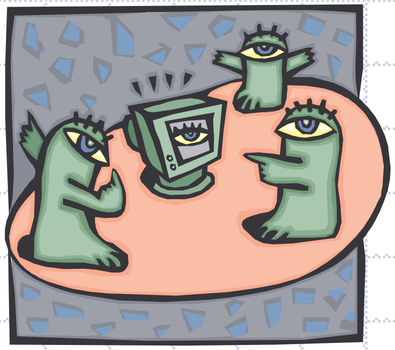


The Proposed Solution

- Create a list of all ms-word files in the network.
- Create a database where the keys are the words and the values are the documents containing this word. We estimated the index will hold 100,000 – 200,000 words, with values containing 1,000 or more documents.
- A web site will provide access to the keyword search using a CGI script.
- We assumed the search will be fast, something like a minute or so… and the indexing will be done every night so it can be slow.

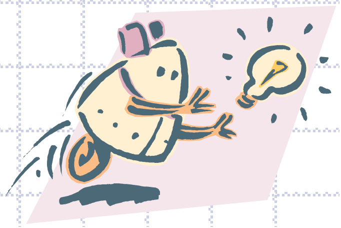
Implementation Options

- Database application using something like FoxPro/Access/… (it is a database after all)
- Custom application using C++ (fast) or VB (easy).
- Using scripting language like Perl or Python (lots of built in features…).



Why Perl?

- The time frame given was small – 10 days.
- ActivePerl for win32 has an easy to use COM support that will automate the conversion of DOC format to some thing manageable…
- I heard that Berkeley DB supported by Perl is excellent choice for the words/documents DB.
- And… there was this article in dr.Dobbs…actually it did what was needed in Perl - there is always someone who did it in Perl before.. And it ***was fast*** – seconds for a search!
- Easy CGI script development.



What Are the Building Blocks?

- File::Find - searching for the files in the network.
- Win32::OLE to automate ms-word to export the documents to HTML format.
- DB file to store the words vs. Documents lists.
- CGI & HTML::Template to create web server search script dynamic pages.
- Internet – to download Tim Kientzle's article sources that reduced the development cycle.

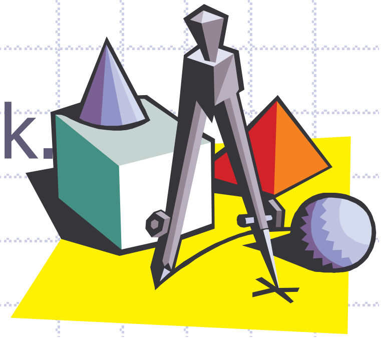
First Step, Collecting Files

- File::Find made it easy:

```
my $FILE_TYPE = "\.doc\$";
find(\&ListFiles_wanted, $ROOT_DIRECTORY);

sub ListFiles_wanted
{
        …
        if($File::Find::name !~ /$FILE_TYPE/i){
                return;
        }
        push @all_files, $File::Find::name ;
}
```

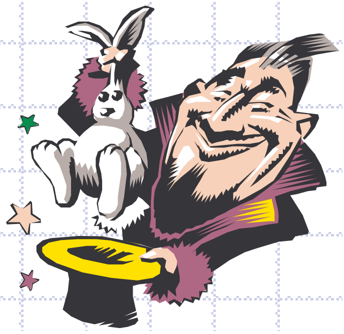
- Security problems are handled gracefully by file::find with no extra work.
- Takes ~two hours.



Second Step – From DOC->HTML

- Save As HTML using Word 2000, yes it works. I did it 30,000 times… ☺
- Win32::OLE has all is needed:

```
$app = Win32::OLE->new('Word.Application');
$app->Documents->Open($docfile, 0, 0, 0,
                      $pswd1, $pswd2, 0,
                      $pswd1, $pswd2);
$doc = $app->ActiveDocument;
$doc->SaveAs($target, $format); #$format = 8;
   (HTML)
$doc->Close;
$app->Quit;
undef $app;
```



Second Step, cont

- It worked for several documents in a loop, but in "field tests" it failed again and again! Locked files, Word closed un-expectedly, and all sorts of other problems…so
  - I put all Word actions in **eval()**.
  - checked for problems and used **Win32::OLE->LastError();**
  - I close word and restart it on every problem, after every "big" file, every 50 files.
  - I try to re-cycle existing instance using **Win32::OLE->GetActiveObject()**

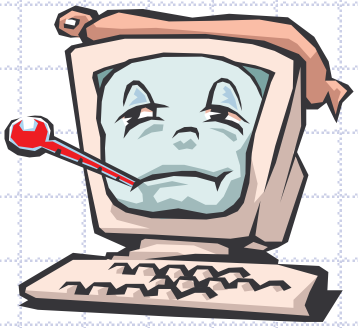
Second Step, conclusion

- Worked for 99.99993% of the documents. (3 documents just would not agree to convert without manual intervention…).
- It takes about 10 hours to convert 30,000 documents.
- Something leaked memory. I find it very hard to find what is leaking so I closed the Perl processes after every 2000 documents (it took five days to debug it to this point).
- The conversion is done only for new files so now it takes something like two hours of file collecting and ~5 seconds per conversion.

Third step – indexing

- All the credit goes to Tim, read the article…
- We open the HTML file and remove all tags and clean off some dust:

```
if (open(HTML_FILE,$indexURL))
{
      local $/;
      $words = <HTML_FILE>;
      close(HTML_FILE);
}
$words =~ s/<[^>]*|>//g; # no more tags
$words =~ s/&nbsp;/ /g;
$words =~ s/[\'\"]//g;  # no more ' "
$words =~ s/\.(\s+)/$1/g; # no more . (1.1)
```



- Lower case every thing too…

Third step, cont

- Get the words list:

```
my(@words) = split(/[^A-Za-z0-9\+\-
                          \.\@\_\$\/\xc0-\xff]+/,$words);
```

- Amazing what a regex may look like… When I need something like this I search regex FAQ lists, there is always something I miss…
- Now remove junk - **grep**:

```
@words = grep { length($_) < 40 } @words;
```

Third step, cont

- Remove duplicates (**precedence**):

```
my(%worduniq); # for unique-ifying word list
@words = grep { $worduniq{$_}++ == 0 } (sort
                                         @words);
```

  - The red phrase above can be hazardous to your health

```
# Every "word" must have at least one alphanumeric
@words = grep { /[a-zA-Z0-9\xc0-\xff]/ } @words;
# Strip out single-character "words"
@words = grep { length > 1 } @words;
```

Third step, cont

- We use Berkeley DB file to store the word index as a binary tree, the keys are the words and the documents ID’s are the values.
- The documents ID’s are the file index in the file list from step one.
- From Perl point of view a DB file looks like a map.

Third step, cont

- The stored information looks like this

  Roey => 10, 12, 3044, 5667, 3000

  Almog => 768, 7657, 4365, 3355

- The we also store the Document name/ID information in the same database

  10 => C:\docs\roey.doc

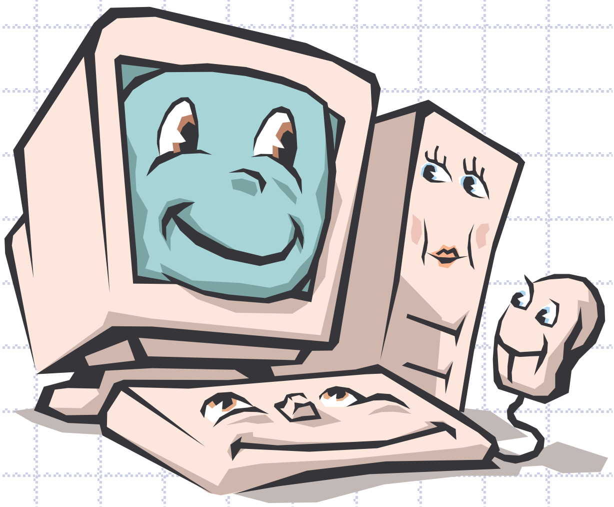
- We save it at the same file by packing the file name with preceding zero.

```
$index{"\0".pack("N",$fileName)} = $fileID;
```

- We use temporary map to cache things to improve speed.

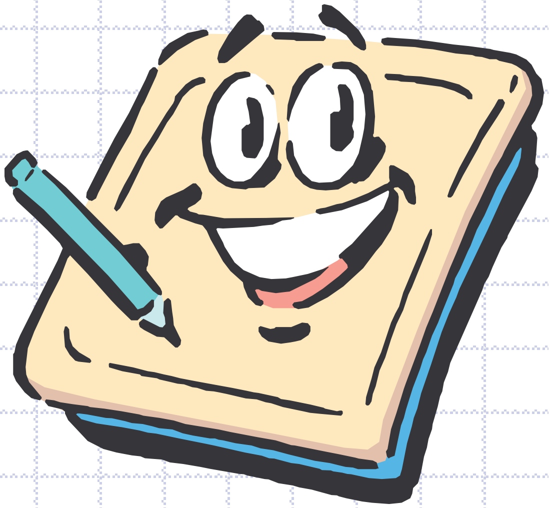
Third step, cont

```
my($wordsIndexed) = 0;
foreach $word (@words) {
        $wordsIndexed++;
        my($a);
        if($wordCache{$word}) {
                $a = $wordCache{$word};
        }
        # use 32 bit unsigned long big indian.
        $a .= pack "N","$fileKey";
        $wordCache{$word} = $a;
}
#%wordCache, sync to disk
if(++$wordCacheCount >= 500) {
   &FlushWordCache();
}
```



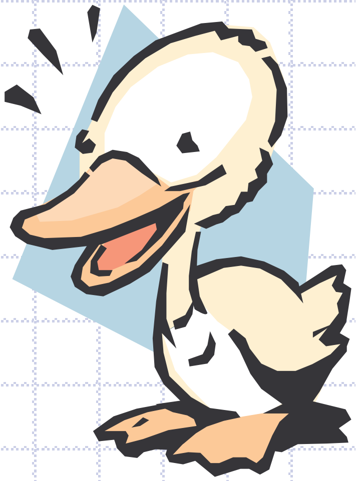
Third Step, cont

- Tim provided some other goodies like
  - Synonyms, you search “Tel Aviv sea” and get “sewage” too…
  - Prevented indexing of common words (the, that, he, it etc…).
  - You search for “מחברת” and get “notebook” too… if you create a dictionary.



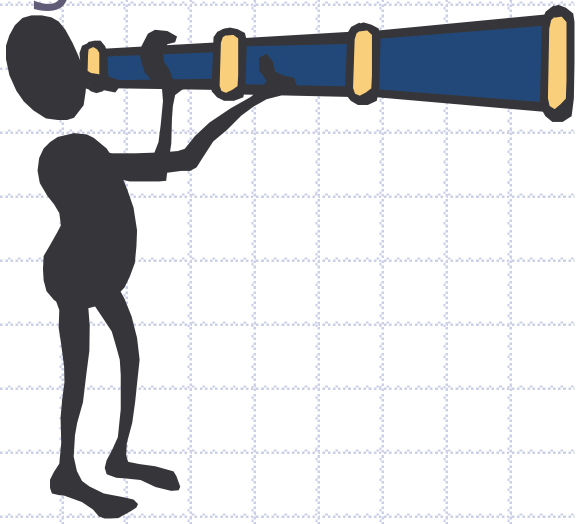
Third step, conclusion

- Tim’s article provide some helpful information regarding Berkeley DB (caching, page sizes etc…)
- All in all most of the indexing was cooked. I did not do much here.
- It worked very well, it took ~10 hours to process all the files in the network for the first time.
- It usually takes 1 - 5 seconds to process HTML file.



Last Step – the search

- For every word search we get a list of results.
- We intersect the the lists.
- Extracts the filenames.
- Display them in a list with using HTML::Template.



Summary

- ***It is <u>impossible</u> to achieve such results in the given time frame using other alternatives***
  - The complete application (indexer & search) worked well in less than 5 partial working days.
  - I wish C++ had something like CPAN.
  - The project was later expanded with new features (phrases search, archive and more…) using Perl.
- Detecting memory leaks?
- Perl is Q&D enabled especially for occasional users like my self.
- Powerful development environment missing?

Links

- Dr. Dobbs Article – (if you are a subscriber)

  http://www.ddj.com/articles/1999/9901/
- Dr. Dobbs Source Code.
  http://www.ddj.com/ftp/1999/1999_01/perlsrch.zip