**[Python toys – I](https://whatha.wordpress.com/2009/02/23/writing-a-search-engine-i/)**

February 23, 2009

This post is the first part of a series of n posts of various toy projects in Python. Python is a very high language and this makes all your work pretty much easy, so that even Tom, Dick and Balaji can write a search engine.

The source code of indexer is as below.  Have a bigger picture.  The explanation for each line of source follows.

#! /usr/sfw/bin/python

import urllib

import re

import pickle

worddict={}

page = "http://www.xkcd.com" #startpage

stack = []

stack.append(page)

visit = {} # keeps track of pages that we visited, to avoid loops

stopvar = 5000

while(stopvar >= 0):

  stopvar-=1

  print stopvar

  cpage = stack.pop(0)

  try:

    f = urllib.urlopen(cpage)

    html = f.read()

  except:

    print cpage,"Unexpected Error"

    continue

  sp = "a href=\""

  htmlvisibletext = re.sub('<.\*?>','',html

  htmlvisibletext = re.sub('[^a-zA-z ]','',htmlvisibletext)

  words=htmlvisibletext.split()

  for word in words:

    word=word.lower()

    if(word not in worddict):

      dict={cpage:1}

      worddict[word]=dict

    else:

      dict=worddict[word];

  for i in range(len(html)):

    if(sp == html[i:i+len(sp)]):

      url = ""

      i+=len(sp)

      while(html[i] != "\""):

        url+=html[i]

        i+=1

      if(url[0:4] == "http"):

        if(visit.has\_key(url) == False):

          visit[url]=1

          stack.append(url)

for key in worddict:

  items = [(v, k) for k, v in worddict[key].items()]

  items.sort()

  items.reverse()

  worddict[key] = [(k, v) for v, k in items]

dumpfile = open('dictdump','wb')

pickle.dump(worddict,dumpfile)

Here We go..

#! /usr/sfw/bin/python

This is the first line similar to most other scripting languages, where you specify the interpreter which will interpret the following script.

import urllib

import re

import pickle

Python provides several modules which we can use. We use here three of those modules

1. urllib – high level interface for fetching data across world wide web

2. re – python’s regular expression module. If you are familiar with Perl’s regex module then this is not much different.

3. pickle – module which provides methods to serialize and store python objects which help in data persistence

 worddict={}

A dictionary data structure in python is used to store pairs of key and an arbitrary object corresponding to the key. worddict is a dictionary we will used to store words and the dictionaries corresponding to the words. Each dictionary corresponding to the word will contain urls in which the word was present and the frequency.

page = "http://www.xkcd.com" #startpage

This is the page where we start our graph traversal.

stack = []

The stack we are using is the python’s list data structure. We initialise to an empty list. This stack will maintain the URLs that we come across in each page.

stack.append(page)

The first page is added to the empty stack.

visit = {} # keeps track of pages that we visited, to avoid loops

stopvar = 5000

This is a limit on the number of pages we want to index. Lets start with a small number 5000. Of course 5000 is a damn small number in this world wide web.

stopvar-=1 #decrement the value

print stopvar

cpage = stack.pop(0)

stack.pop() can be used to pop values. stack.pop() pops the last element appended. Stack.pop(0) pops(??) the 1st element in the list. We are taking the first element in the list. So this is Basically a DFS.

 try:

    f = urllib.urlopen(cpage)

    html = f.read()

The try-except construct is used to handle exceptions. Any exceptions in try block are thrown to the except block and are handled there.

except:

print cpage,"Unexpected Error"

continue

The exception is handled with the just the continue command.

sp = "a href=\""

“a href = “”  is the pattern we will search in the html page to capture urls.

htmlvisibletext = re.sub('<.\*?>','',html

htmlvisibletext = re.sub('[^a-zA-z ]','',htmlvisibletext)

words=htmlvisibletext.split()

for word in words:

word=word.lower()

We convert all words to lower case to provide for case-insensitivity.

 if(word not in worddict):

      dict={cpage:1}

      worddict[word]=dict

If the word is not already present in the worddict dictionary, we create a dictionary dict with contents current url and frequency 1. The word and the dictionary dict are then inserted into the worddict.

else:

   dict=worddict[word];

In case the word is already there we obtain the dictionary by looking up for the word.

     if(cpage not in dict):

        dict[cpage]=1

      else:

        dict[cpage]=dict[cpage]+1

Now we check if the url is present in the dictionary for that particular word. Suppose the word occurs for the first time in a url, cpage will not be present in dict. so url and corresponding frequency 1 are added to the dictionary.

If the word was already present in the url and is repeated, then the frequency is incremented by 1.

 for i in range(len(html)):

    if(sp == html[i:i+len(sp)]):

      url = “”

      i+=len(sp)

the html is a string which contains the whole html file corresponding to the url. Now we scan it to find <a href=. If we come across <a href = “, we store the consequent string in url until we come across “.

      while(html[i] != “\””):

        url+=html[i]

        i+=1

      if(url[0:4] == “http”):

        if(visit.has\_key(url) == False):

          visit[url]=1

          stack.append(url)

if the url starts with http, we check if the visit dictionary, which holds all the urls that we ve visited. If we have not visited, we add the url to our dictionary and append it to our stack.

for key in worddict:

  items = [(v, k) for k, v in worddict[key].items()]

  items.sort()

  items.reverse() # so that highest value is first

  worddict[key] = [(k, v) for v, k in items]

The above is the general mechanism used to sort a dictionary by key. We sort the dictionary corresponding to each word so that the url with the highest frequency comes to the top.

  dumpfile = open(‘dictdump’,’wb’)

I open a file named ‘dumpfile’ in the current directory for writing. We will dump our index into this file.

  pickle.dump(worddict,dumpfile)

Here we use the dump function provided by the pickle module to dump the object into the file.

Now that we have dumped it, Should we do wat we started of promising to. Lets now write the script to search the word from the dump.

#! /usr/sfw/bin/python

import pickle

dumpfile=open(‘dictdump’,’rb’)

worddict=pickle.load(dumpfile)

print ‘Press CTRL + C to kill’

while 1:

  word=raw\_input(‘Enter word to search:’)

  word=word.lower()

  if(word not in worddict):

    print ‘Word not found’

  else:

    print worddict[word]

Now we ll look into it closely.

import pickle

We again import the pickle module to load the dumped dictionary

dumpfile=open(‘dictdump’,’rb’)

worddict=pickle.load(dumpfile)

Open the dumpfile in readmode and load the dictionary.

word=word.lower()

Convert the query to lower case.

if(word not in worddict):

    print ‘Word not found’

  else:

    print worddict[word]

Print the dictionary corresponding to the word.