Research Project 2:
Roll Your Own Mini Search Engine

刘尚青 3150105397 漆翔宇 3170104557 卢雨洁 3150105267

2018-03-25

Chapter 1: Introduction

1.1 Problem Description

Our group aimed at implementing a mini search engine to provide the users the function of inquiring over the text on the internet.

1.2 Purpose

The report contains the idea of our group about how to implement a mini search engine. The algorithms that we used to finish the project would also be illustrated in this report. To help others have a better understanding of our design, we not only display the source code in the appendix but also the pseudo-code in the second part.

1.3 Background

The main algorithm in our spider is BFS which is a blind search method that aims to systematically expand and examine all nodes in the graph to find the results. And we used splay tree, Red-Black tree and AVL tree to store our inverted-index structure.

Chapter 2: Data Structure / Algorithm Specification

2.1 Algorithms

Our steps to implement the mini search engine is that implement the spider to scratch the pages we need and then scan the text and count out the words in the texts. Use the implemented function to finish word stemming and eliminate the stop words. After that we built the inverted index structure based on the left words.

2.1.1 Spider

We implemented the spider to crawl the web data.

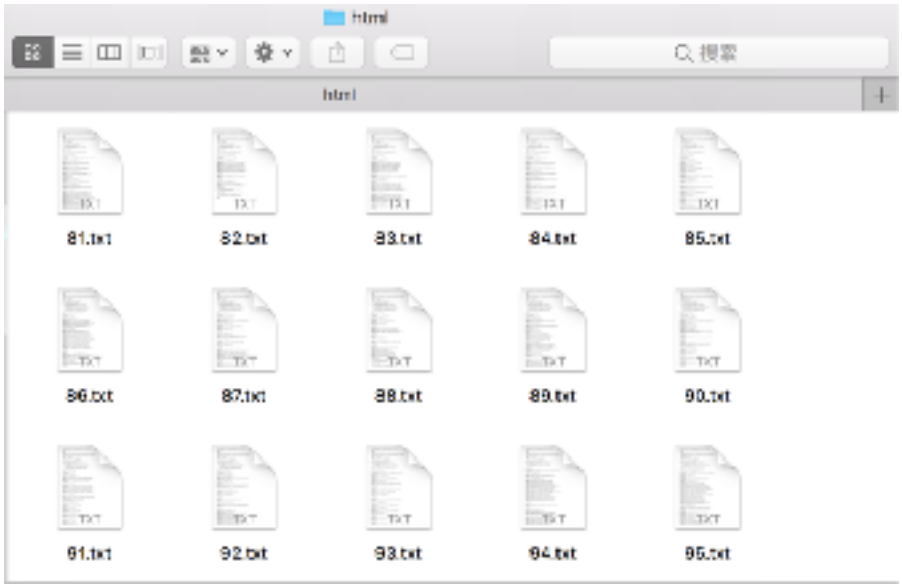
We first construct the data structure of the URL to store the root, the next and the level of current URL.

At the beginning, we push the frontage into the queue. And then we use the algorithm of Breadth-First-Search to access the page at the front of the queue.

When we access one item, we would record the page's information, and get it's name. The page's address would be stored in the loc.

And then we use the function URLDownloadToFile to download the page and print out the page's url we download to ensure that we've downloaded the exact pages we want.

Here is a part of the picture of the pages we've downloaded through our spider.



Here is a piece of pseudo-code of our spider based on C programming language.

```
struct URL{…}
CreateDirectory for pages
//initialize file root
file_root = (“./html//”)
queue<URL>queue;
queue.push(frontpage)
while( queue is not empty)
{
   temp = queue.front()
   pop the top item of the queue
   analysis()
}
```

In the function of analysis(), we use BFS to go over each page.

2.1.2 ETreeParse

We implemented the ETreeParse to parse the text into the pure content of the text.

```
from lxml import etree
for i in range(45,992):
    temp = i-39
    myhtml = etree.parse('./html/'+str(i)+'.txt',etree.HTMLParser())
    title = myhtml.xpath('//title/text()')
    print title
    newtitle = [i.replace('\n',' ')for i in title]
    print newtitle
    result = myhtml.xpath('//td/text()|//p/text()|//h1/text()|//h2/text()|//h3/t
    print result
   # h1 = myhtml.xpath('//h1/text()')
    #print h1
   # result = myhtml.xpath('//body//blockquote/text()')
    #print result
    f = open('data//'+str(temp)+'.txt','w')

    for t in newtitle:
        f.write(t+'\n')
    for line in result:
        f.write(line)
    f.close()
```

2.1.3 Inverted

We implemented the Inverted code part to eliminate the stop words and then using porterStemming to finish the word stemming work.

At last we finished building inverted-index structure.

```
#-*- coding:utf-8 -*-
import os
import json
import time
dict = {}
lineNum = 0
offset = 0
start = time.time()
sum = 0.0
for pname in os.listdir('pdata'):
    f = open('pdata//'+pname,'r')
    fileindex = pname[:-4]
    for line in f:
        lineNum = lineNum + 1
        offset = 0
        t = line.split()
        for word in t:
            offset = offset + 1
            if word not in dict:
                item = [fileindex,lineNum,offset]
                index = []
                index.append(item)
                dict[word] = index
            else:
                item = [fileindex,lineNum,offset]
                dict[word].append(item)
    f.close()
with open("index.json",'w') as f:
    json.dump(dict,f,indent=4)
end = time.time()
sum = sum + end - start
print sum
```

2.2 Data Structures

The main data Red-Black Tree , AVL Tree, Splay Tree data structures and the URL structure in the spider program are the main data structures in our project.

2.2.1 Spider Structure

The struct of the URL in our spider

```
struct URL
{
        string root, next;
        int level;
}temp;
```

2.2.2 RBTree / AVLTree / SplayTree

RBTree data structure

```
struct RBtree
{
    int color,size;
    string word, filename;
    int lineNum;
    int Offset;
    RBtree *ls,*rs,*fz;
    RBtree(int hue,string _word,string _name,int _num,int
    _set,RBtree *anc,RBtree *l,RBtree *r,int s):
    color(hue),word(_word),filename(_name),lineNum(_num),
    Offset(_set),fz(anc),ls(l),rs(r),size(s){};
}*nil=new RBtree(black,"","","",0,0,0,0),*root=nil;
```

AVLTree data structure

```
struct AVL
{
    int height;
    AVL *ls,*rs,*fz;
    string word,filename;
    int lineNum, offset;
    Splay(Splay *father,string _word,string _name,string
    _set)):ls(0),rs(0),fz(father),size(1),word(_word),
    filename(_name),fileOffset(_set){};
    AVL(string _word,string _name,int _num,int _set,int h,
    AVL *f,AVL *l,AVL *r):
    word(_word),filename(_name),lineNum(_num),offset(_set)
    ,height(h),fz(f),ls(l),rs(r){};//initialization
}*_none=new AVL("","","",-1,0,0,0),*root=_none;//define
```

SplayTree data structure

```
struct Splay
{
    Splay *ls,*rs,*fz;
    int size;
    string word,filename,fileOffset;
    Splay(Splay *father,string _word,string _name,string
    _set]):ls(0),rs(0),fz(father),size(1),word(_word),
    filename(_name),fileOffset(_set){};
}*root=0,*_none=new Splay(0,"","","");//define the Splay
```

We implemented the data structure of the Red-Black Tree, AVL Tree, Splay Tree to store the structure of inverted index.

Chapter 3: Testing Results

3.1 Speed of Index

Here is the experimental testing results of our inverted-index structure.

Inverted count
12.0360000134
11.6659998894
22.4200000763
20.5450000763
23.1889998913
20.7679998875
22.9120001793
20.3289999962
21.3619999886
18.9139997959

The result is based on the 10 times of test.

In python I use the time library to count the time which I did it for 10 times and calculate the average of it.

We fount out that using time library has problem when the pycharm idle run in many thread, it would cost time on other operation which increase the time level so I could only test it by order and make average afterward.

3.2 Speed of Search

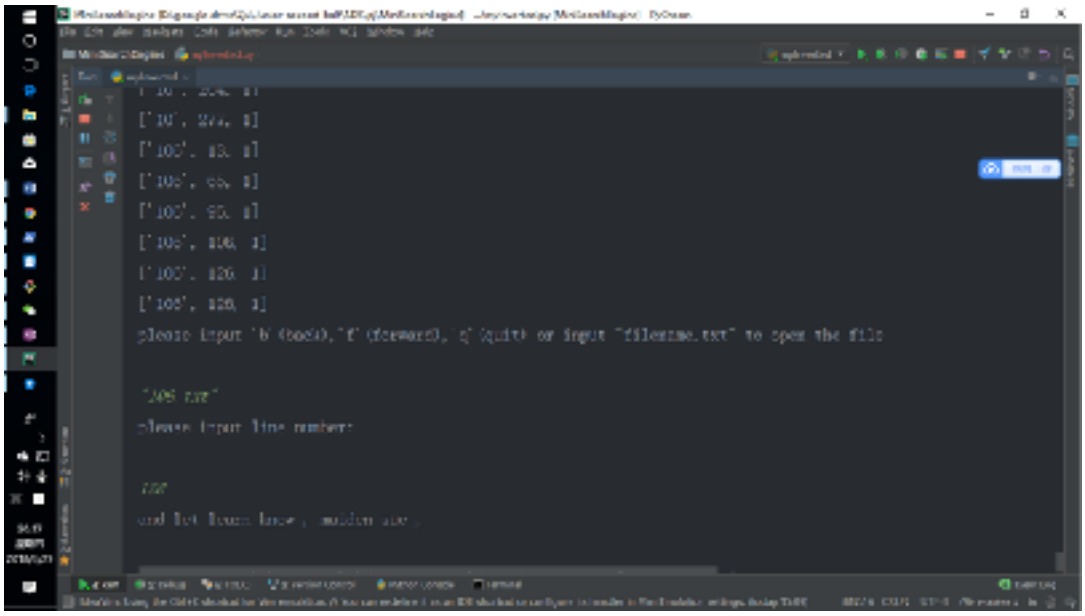
We test couple words for timing like "and" "clamour" "sometim"... and the average of time is 0.365999937057 sec

And this is the structure of the directly search in python dict

3.3 Result of Search

This is the threshold of query.

Each time visit the line of the file. The threshold of the specific file would minus one. Every time in query would need to find the min() of five file name and output that name



Threshold Search Time

test time1:
3.91799998283
test time2:
3.33799982071
test time3:
2.79199981689
test time4:
3.40900015831
test time5:
2.367000103

Average Time of Simple Search

0.280999898911

Chapter 4: Analysis and Comments

4.1 Complexity

About time complexity

The spider is bfs O(|V| + |E|)

Inverted index build is O(n^n), because we search every word for all the file and make out the index.

The search is using dictionary O(n)

We plan to use other data structure to save the inverted index like balance tree and it would turn the time complexity to O(logn) which would have better performance.

4.2 Compare

After analyzing several pages, we notice that the relationship among the pages in the website can be simplified as a tree, which means we can use few particular judges to avoid accessing a page again rather than using hash or set to make it.

So we define the type URL to memorize a url and it's depth.The url is divided into 2 part "root"and"next" which will make the implement easier.

4.3 Possible Improvement

Using term frequency to judge the relativity between the key words and pages.

Let TFi be the term frequency of key word i; And to make this way efficient, we should delete the stop words from the key words. Then we make TF1+.....+TFN be the relativity between our N key words and the pages.

But it's also not perfect enough. What's more, if a key word’s appearance frequency in all the pages is lower than another particular key word.Sure, we should give it a higher right.

If a word w appears in Dw pages. Then, the larger Dw is, the lower the right of w should be.

So we use the formula IDF=log(D/Dw) in which D is the number of all the pages to define a key
word w'right IDw.

So, we can finally use TF1*IDF1+TF2*IDF2+.....+TFN*IDFN to be the relativity.

Obviously, we can set a thresholding, if the relativity is lower than it, we don't need to show it in our search results. And we can use the relativity as a key to sort the result.

Appendix: Source Code

spider.cpp

```
#include <windows.h>
#include <iostream>
#include <fstream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <sstream>
#include <set>
#include <queue>
#include <UrlMon.h>
#pragma comment(lib, "urlmon.lib")
using namespace std;
/*
As we use the system function "URLDownloadToFile" to download the website,
wo will meet the situation where we need to transfer string into LPCSWTR.
The function stringToLPCWSTR will do this.
*/
LPCWSTR stringToLPCWSTR(std::string orig)
{
    size_t origsize = orig.length() + 1;
    const size_t newsize = 100;
    size_t convertedChars = 0;
    wchar_t *wcstring = (wchar_t *)malloc(sizeof(wchar_t)*(orig.length() - 1));
    mbstowcs_s(&convertedChars, wcstring, origsize, orig.c_str(),
_TRUNCATE);

    return wcstring;
}
```

```
ifstream infile(loc);
        string TEXT; char t;
        while ((t = infile.get()) != EOF)TEXT = TEXT + t;
        //use a string TEXT to read the page we access.
        const char *pos = TEXT.c_str();//define a pointer of it.
        while (pos != NULL)//capture the url in this page.
        {
            pos = strstr(pos, "href=\"");//find the next url.
            if (pos != NULL)//if we found it,then get it.
            {
                pos += strlen("href=\"");
                //We notice that the url begins with "http" would point outside of
                //the website.No need to get them.
                if (*pos == '/'||(*pos ==
'h'&&*(pos+1)=='t'&&*(pos+2)=='t'&&*(pos+3)=='p'))continue;
                else
                {
                    temp.root=rt;
                    temp.next.clear();
                    while (*pos != '/' && *pos != '\"')
                    {
                        temp.root = temp.root + (*pos);
                        pos++;

                    }//get the url's first part;
                    if (lev == 1)
                    {
                        /*
                          as only the page in the level 1 and 2 will have
                          a the second part,so only when we capture the url in
                          level 1 we need to get it.
                        */
if (*pos == '/')
                        {
                            temp.root = temp.root + '/';
```

```
while (*pos != '\"')
                    {
                        temp.next = temp.next + (*pos);
                        pos++;
                    }
                }
            }
            //if (cnt == 39){ cout << "aaaaaaaaaaaaaaaaaaaaaa!!!!!!!!!!" <<
endl; }
            temp.level = lev + 1;
            que.push(temp);//insert the url captured into the queue.
        }
    }

    }
    pos = TEXT.c_str();
    while (pos != NULL)//capture the url in this page.
    {
        pos = strstr(pos, "HREF=\"");//find the next url.
        if (pos != NULL)//if we found it,then get it.
        {
            pos += strlen("HREF=\"");
            //We notice that the url begins with "http" would point outside of
            //the website.No need to get them.
            if (*pos == '/' || (*pos == 'h'&&*(pos + 1) == 't'&&*(pos + 2) ==
't'&&*(pos + 3) == 'p'))continue;
            else
            {
                temp.root = rt;
                temp.next.clear();
                while (*pos != '/' && *pos != '\"')
                {
                    temp.root = temp.root + (*pos);
                    pos++;
```

```
                    }
                }
            }
            //if (cnt == 39){ cout << "aaaaaaaaaaaaaaaaaaaaaa!!!!!!!!!!" <<
endl; }
            temp.level = lev + 1;
            que.push(temp);//insert the url captured into the queue.
        }
    }
  }
}
int main()
{
  CreateDirectory(stringToLPCWSTR("./html"), 0);//create a directory for
the pages.

  file_root = ("./html//");
  temp.root = rt;
  temp.next = nt;
  temp.level = lev;

  que.push(temp);//push the frontpage into the queue.
  while (!que.empty())//BFS.
  {
    temp = que.front();
    que.pop();
    analysis();
  }
  getchar();
  return 0;
}
```

```
/*

After analysing several pages,we notice that the relationship among the pages in the website can
be simplified as a tree,which means we can use few particular judges to avoid accessing a page again
rather than using hash or set to make it.

So we define the type URL to memorize a url and it's depth.
the url is devided into 2 part "root"and"next" which will make the implement easier.

*/
struct URL
{
    string root, next;
    int level;
}temp;
queue<URL>que;
/*
    We use BFS to go over each page.
    que is for BFS.
*/

int cnt = 0;
/*
    We simply use the order we access a page as the page's name.
    So we use cnt to record it.
*/
string rt = "http://shakespeare.mit.edu/";
string nt = "index.html";
int lev = 1;
//rt nt and lev is used as a temporary variables.
//initially we let them record the frontpage's information.
```

```
string file_root;
string file_name;
//use file_root to record the place we store the pages we download.
//use file_name to record the name of the new page waited to be
downloaded.
string getid() //get the name of a new file.
{
    cnt++;
    stringstream  t;
    t<<cnt;
    return (t.str()+".txt");
}
void analysis() //access the page at the front of the BFS que.
{
    rt = temp.root;
    nt = temp.next;
    lev = temp.level;//record the page's information.
    file_name = getid();//get it's name
    string target = rt + nt;  string loc = file_root + file_name;
    //target is the page's address  and  loc is the place we will store it.
    LPCWSTR str1 = stringToLPCWSTR(target);
    LPCWSTR str2 = stringToLPCWSTR(loc);//transfer string into LPCWSTR.
    URLDownloadToFile(NULL, str1, str2, 0, 0);//download the page.

    cout << cnt << " " << target << endl;//print the page's url we download.

    if (lev == 3)return;
    /*
        we notice that the url in the page of depth 3 point
        either it's ancestors or it's brothers.
        no need to analyze it.
    */
    else
    {
```

myEtreeParse.py

```
from lxml import etree
for i in range(45,992):
    temp = i-39
    myhtml = etree.parse('./html/'+str(i)+'.txt',etree.HTMLParser())
    title = myhtml.xpath('//title/text()')
    print title
    newtitle = [i.replace('\n',' ')for i in title]
    print newtitle
    result = myhtml.xpath('//td/text()|//p/text()|//h1/text()|//h2/text()|//h3/text()|//
blockquote/i/text()|//blockquote/a/text()|//a/b/text()|//p/i/text()|//address/
text()|//td/a/text()|//body/blockquote/text()')
    print result
   # h1 = myhtml.xpath('//h1/text()')
    #print h1
   # result = myhtml.xpath('//body//blockquote/text()')
    #print result
    f = open('data//'+str(temp)+'.txt','w')

    for t in newtitle:
        f.write(t+'\n')
    for line in result:
        f.write(line)
    f.close()
```

myInvertedTimeTest.py

```
#-*- coding:utf-8 -*-
import os
import json
import time
dict = {}
lineNum = 0
offset = 0
start = time.time()
sum = 0.0
for pname in os.listdir('pdata'):
    f = open('pdata//'+pname,'r')
    fileindex = pname[:-4]
    for line in f:
        lineNum = lineNum + 1
        offset = 0
        t = line.split()
        for word in t:
            offset = offset + 1
            if word not in dict:
                item = [fileindex,lineNum,offset]
                index = []
                index.append(item)
                dict[word] = index
            else:
                item = [fileindex,lineNum,offset]
                dict[word].append(item)
    f.close()
with open("index.json",'w') as f:
    json.dump(dict,f,indent=4)
end = time.time()
sum = sum + end - start
print sum
```

```
void splay(Splay *x)//splay operations.
{
    Splay *y,*z;
    while(x->fz!=_none)//if x!=root ,it should go on being rotated up
    {
        y=x->fz;z=y->fz;
        if(z!=_none)//if y is not the root,we would use doulbe rotaion
        {

            if(z->ls==y)
            {
                if(y->ls==x){zig(y);zig(x);}
                else {zag(x);zig(x);}
            }
            else
            {
                if(y->rs==x){zag(y);zag(x);}
                else {zig(x);zag(x);}
            }
        }
        else //if y is the root , use single rotaion
        {
            if(y->ls==x)zig(x);else zag(x);
        }
        //the detailed rotaion rules have been declared in the project report.
    }
    root=x;//x has become the root.
}
Splay *find(string x)//alone the tree to find the vertex we want.
{
    Splay *p=root;
    while(p)
    {
        if(p->word==x)break;
```

```
if(x<p->word)p=p->ls;
        else p=p->rs;
    }
    return p;
}
void _insert(string _word,string _name,string _set)
{
    if(root==0)//if the tree is empty,make the new element the root.
    {
        root=new Splay(_none,_word,_name,_set);
        return;
    }
    else
    {
        //start from root ,find the place first, the same to AVL code.
        Splay *p=root;
        while(p)
        {
            if(x<p->word)
            {
                if(p->ls)p=p->ls;
else
                {
                    Splay *temp=new Splay(p,_word,_name,_set);
                    p->ls=temp;
                    splay(temp);  return;
                }
            }
            else
            {
                if(p->rs)p=p->rs;
                else
                {
                    Splay *temp=new Splay(p,_word,_name,_set);
```

```
p->rs=temp;
                splay(temp);return;
            }
        }
    }
}
Splay *getmax(Splay *x)// get the max element in the sub-tree x.
{
    if(!x)return x;
    while(x->rs)x=x->rs;
    return x;
}
Splay *getmin(Splay *x)// get the min element in the sub-tree x.
{
    if(!x)return x;
    while(x->ls)x=x->ls;
    return x;
}
void _del(string x)//del in Splay is very simple.
{
    Splay *p=find(x);  //find the point to be deleted.
    if(p==0)cout<<"error!"<<endl;
    if(p)//if the point exist,del it.
    {
        splay(p);//first make it the root .
        Splay *L=p->ls,*R=p->rs;delete p; //delete it directly (much more direct
than that in AVL or BST)
        if(L==0){root=R;if(R)R->fz=_none;return;}//if it's left-subtree is empty,
make the roof of the right-subtree the root.
        else
        {
            root=L;root->fz=_none;
            //make left-subree's max element the root of the tree,thus that the
```

myEtreeParse.py

```
from lxml import etree
for i in range(45,992):
    temp = i-39
    myhtml = etree.parse('./html/'+str(i)+'.txt',etree.HTMLParser())
    title = myhtml.xpath('//title/text()')
    print title
    newtitle = [i.replace('\n',' ')for i in title]
    print newtitle
    result = myhtml.xpath('//td/text()|//p/text()|//h1/text()|//h2/text()|//h3/text()|//
blockquote/i/text()|//blockquote/a/text()|//a/b/text()|//p/i/text()|//address/
text()|//td/a/text()|//body/blockquote/text()')
    print result
   # h1 = myhtml.xpath('//h1/text()')
    #print h1
   # result = myhtml.xpath('//body//blockquote/text()')
    #print result
    f = open('data//'+str(temp)+'.txt','w')

    for t in newtitle:
        f.write(t+'\n')
    for line in result:
        f.write(line)
    f.close()
```

Splay.cpp

```
struct Splay
{
    Splay *ls,*rs,*fz;
    int size;
    string word,filename,fileOffset;
    Splay(Splay *father,string _word,string _name,string
_set)):ls(0),rs(0),fz(father),size(1),word(_word),filename(_name),fileOffset(_
set){};
}*root=0,*_none=new Splay(0,"","","");//define the Splay node, the usage of
_none is the same to that in the AVL code
void zig(Splay *x)//zig and zag is the same to those in the AVL code,no more
comment.
{
    Splay *y=x->fz,*z=y->fz;
    if(y==z->ls)z->ls=x;else z->rs=x;
    x->fz=z;y->ls=x->rs;
    if(y->ls)y->ls->fz=y;
    x->rs=y;y->fz=x;
    x->size=y->size;
    y->size=(y->ls?y->ls->size:0)+(y->rs?y->rs->size:0)+1;
}
void zag(Splay *x)//zig and zag is the same to those in the AVL code,no
more comment.
{
    Splay *y=x->fz,*z=y->fz;
    if(y==z->ls)z->ls=x;else z->rs=x;
    x->fz=z;y->rs=x->ls;if(y->rs)y->rs->fz=y;
    x->ls=y;y->fz=x;
    x->size=y->size;
    y->size=(y->ls?y->ls->size:0)+(y->rs?y->rs->size:0)+1;
}
```

```
no right child
            //so we can directly make the right sub-tree of the origional tree the
right son of the left-subtree.
            Splay *temp=getmax(L);
            splay(temp);
            if(R)//update the information.
            {
              root->rs=R;R->fz=root;
              root->size+=R->size;
            }
            return;
        }
    }
}
```

The other two piece of source code would be in the final zip package. We display the splay tree source code but not the red-black tree and AVL tree source code.

References

[1]吴军，“数学之美”，人民邮电出版社，2012-5

Author List

Tester : Liu Shangqing
Coder : Qi Xiangyu
Writer : Lu Yujie

Declaration

We hereby declare that all the work done in this project titled "Roll Your Own Mini Search Engine" is of our independent effort as a group.