

实验报告

实验 2 & 3
517030910374
郭嘉宋

1. 实验准备

(1) 实验环境

本实验采用 Ubuntu 系统下的 python2.7 进行，使用 miniconda 环境，Ubuntu 系统安装在 VMware Workstation 提供的虚拟机中，同时使用 BeautifulSoup、urllib2 等库对网页进行解析。

(2) 实验目的

实验 2：

了解 http 协议的工作原理，学习爬虫的使用方法，编写程序在网页上寻找需要的 URL，了解 get 和 post 两种请求网页方式。了解深度优先和广度优先两种搜索方式，掌握其搜索原理，用 python 模拟两种请求方式和搜索方式。

实验 3：

使用 hash 函数简化 list 每次遍历带来的时间损耗，学会编写自己的 bloomfilter，优化遍历时的时间消耗，优化 bloomfilter。学会使用并发编程，同时使用 varlock 保护全局变量，优化爬虫的效率。

(3) 实验原理

实验 2：

1. 使用 python 模拟登陆交大 BBS，使用谷歌浏览器找到相应参数，发出请求，修改其个人签名档，输入对应参数即可。
2. 简单的对数组进行插入操作即可。
3. bfs 操作只需将新的链接添加至队伍末尾，即可实现。图的结构只需每一次将关联地址添加到字典中最后统一输出 graph 即可。
4. 修改原来 g 中图的结构使用 urllib2 爬取网页即可。

实验 3：

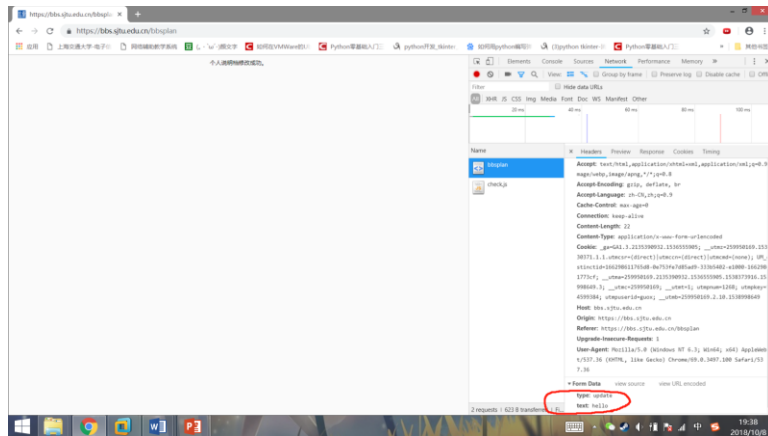
5. 自己编写 bloomfilter，使用随机生成的字符串输入 bloomfilter，同时使用随机生成的字符串对错误率进行检验。使用 bitarray 类完成 bloomfilter，用公式 $\ln 2 * m/n$ 计算出最佳的哈希函数个数一一映射即可完成。
6. 将 seed 网页和所有待爬网页加入 queue 队列，threading.Thread 使用多线程进行爬虫，再使用 varLock 对全局变量进行保护，即可优化爬虫效率。

2. 实验过程

(1) 练习 1

1.1 实验步骤

先使用图示方法登录交大 BBS，我使用了自己注册的账号和密码进行登录。之后再使用 google 浏览器中查看到原网页的两个参数的值，见下图右下角红圈，使用 python 模拟发出请求即可修改。



```
def bbs_set(id, pw, text):
    import urllib2, cookielib, urllib
    from bs4 import BeautifulSoup
    cj = cookielib.CookieJar()
    opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
    urllib2.install_opener(opener)
    postdata = urllib.urlencode({
        'id': id,
        'pw': pw,
        'submit': 'login'
    })

    new_data = urllib.urlencode({
        'type': 'update',
        'text': text
    })
```

```
print new_data
req1 = urllib2.Request(url='https://bbs.sjtu.edu.cn/bbslogin', data=postdata)
urllib2.urlopen(req1)
req2= urllib2.Request(url='https://bbs.sjtu.edu.cn/bbsplan', data=new_data)
req3= urllib2.Request(url='https://bbs.sjtu.edu.cn/bbsplan')

urllib2.urlopen(req2).read()
content1 = urllib2.urlopen(req3).read()

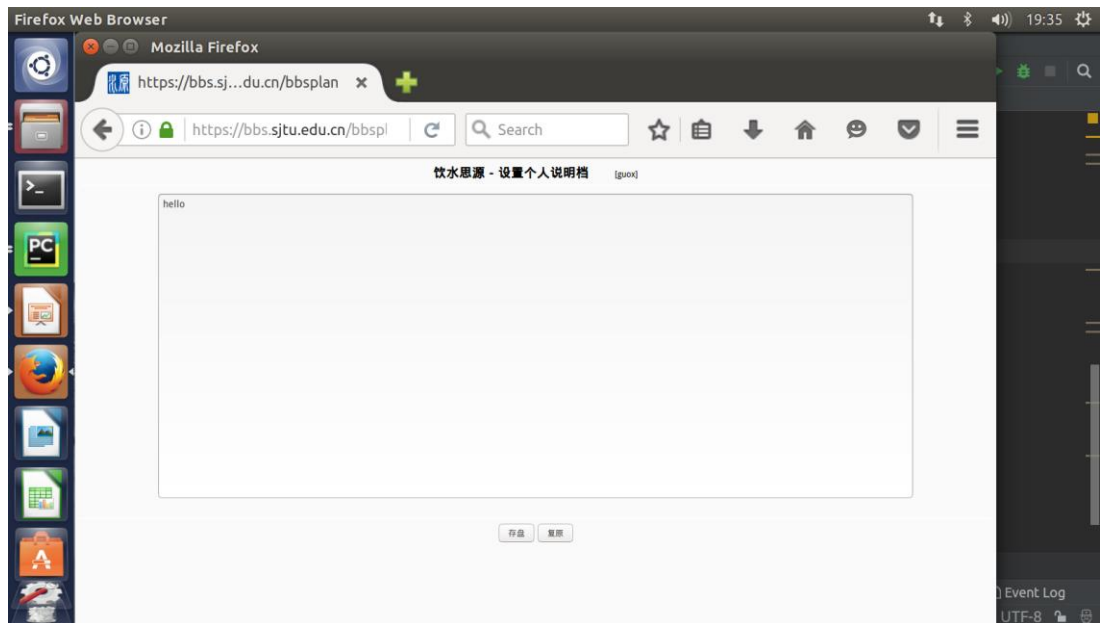
soup = BeautifulSoup(content1)
print str(soup.find('textarea').string).strip().encode('utf8')
```

NOTE：之前一直在网页的 submit 按钮附近查找对应参数，实则应在 network 页面下查找提出请求时的对应参数 update。

1.2 实验结果

运行了上述程序后，即可更改网页内容，如图，修改成为所演示的 hello。

```
network1.py
guo@ubuntu: ~/PycharmProjects/lab2
guo@ubuntu:~$ cd PycharmProjects
guo@ubuntu:~/PycharmProjects$ cd lab2
guo@ubuntu:~/PycharmProjects/lab2$ python homework1.py guox 000000 hello
text=hello&type=update
```



NOTE :第一次实验时一直更改不了,后来得知原因 Request 仅在本地图包好并未上传, 因此登录时的 urlopen 不可省略, 之前省略了上传的步骤, 故每次 request 皆要上传本地的请求内容。

(2) 练习 2

2.1 实验步骤

练习 2 仅需对数组进行操作, 将两数组合并, 较容易, 详见代码即可。

2.2 实验结果

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
lab2 ~/PycharmProjects/lab2
Project
lab2 ~/PycharmProjects/lab2
html
bbs_set_sample.py
crawler.py
crawler_sample.py
hello.py
homework1.py
homework2.py
homework3.py
homework4.py
index.txt
test.py
External Libraries
Run: homework2
/home/guo/miniconda2/bin/python /home/guo/PycharmProjects/lab2/homework2.py
[5, 4, 1, 2, 3] [2, 4, 4, 5]
Process finished with exit code 0
4: Run TODO Terminal Python Console
Event Log
5:1 LF : UTF-8
```

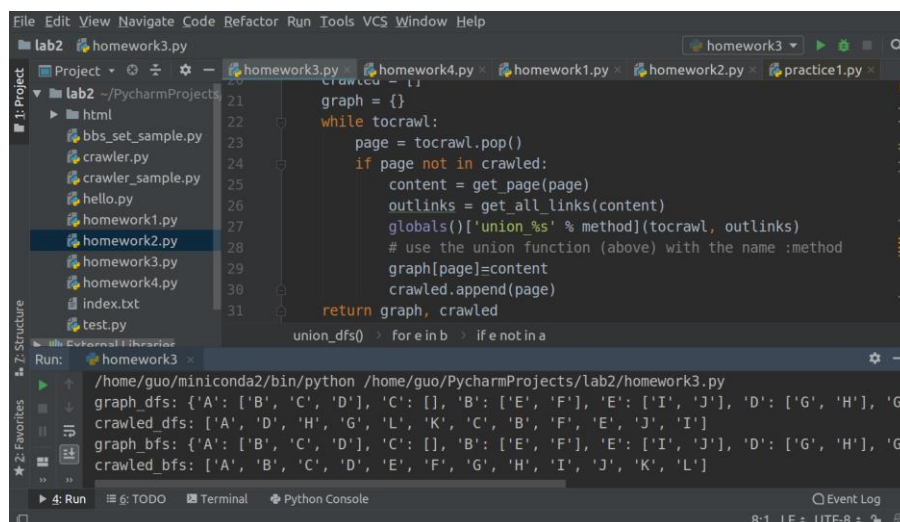
Python 输出的即为两个新数组。

(3) 练习 3

3.1 实验步骤

Dfs 已经给出了相应算法，Bfs 需要更改的仅是将新爬到的 url 放到队列的尾，用练习 2 的 union_bfs 取代之前的 union_dfs 函数即可。图的结构只需要每次向字典添加元素即可，较容易，详见代码。

3.2 实验结果



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
lab2 ~\PycharmProjects\lab2\homework3.py
21 graph = {}
22 while tocrawl:
23     page = tocrawl.pop()
24     if page not in crawled:
25         content = get_page(page)
26         outlinks = get_all_links(content)
27         globals()['union_%s' % method](tocrawl, outlinks)
28         # use the union function (above) with the name :method
29         graph[page]=content
30         crawled.append(page)
31     return graph, crawled
union_dfs() for e in b if e not in a

Run: homework3
/home/guo/miniconda2/bin/python /home/guo/PycharmProjects/lab2/homework3.py
graph_dfs: {'A': ['B', 'C', 'D'], 'C': [], 'B': ['E', 'F'], 'E': ['I', 'J'], 'D': ['G', 'H'], 'G': ['K'], 'H': ['L'], 'I': ['J'], 'J': ['I']}
crawled_dfs: ['A', 'D', 'H', 'G', 'L', 'K', 'C', 'B', 'F', 'E', 'J', 'I']
graph_bfs: {'A': ['B', 'C', 'D'], 'C': [], 'B': ['E', 'F'], 'E': ['I', 'J'], 'D': ['G', 'H'], 'G': ['K'], 'H': ['L'], 'I': ['J'], 'J': ['I']}
crawled_bfs: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L']
```

NOTE : 添加 graph[page]=content 即可得到图。

(4) 练习 4

4.1 实验步骤

修改程序中对应的 get_page() 函数，使其能够根据给定的 url 自动爬取其中复函的 url，使用 try 和 except 的方式防止网页无法打开，同时在 urlopen 时添加超时参数 timeout 防止网页超时。Seed 为爬虫的种子，使用 bfs 和 dfs 两种方式从 seed 开始查找，方便起见，max_page 我选取了 10，实验操作详见下图。

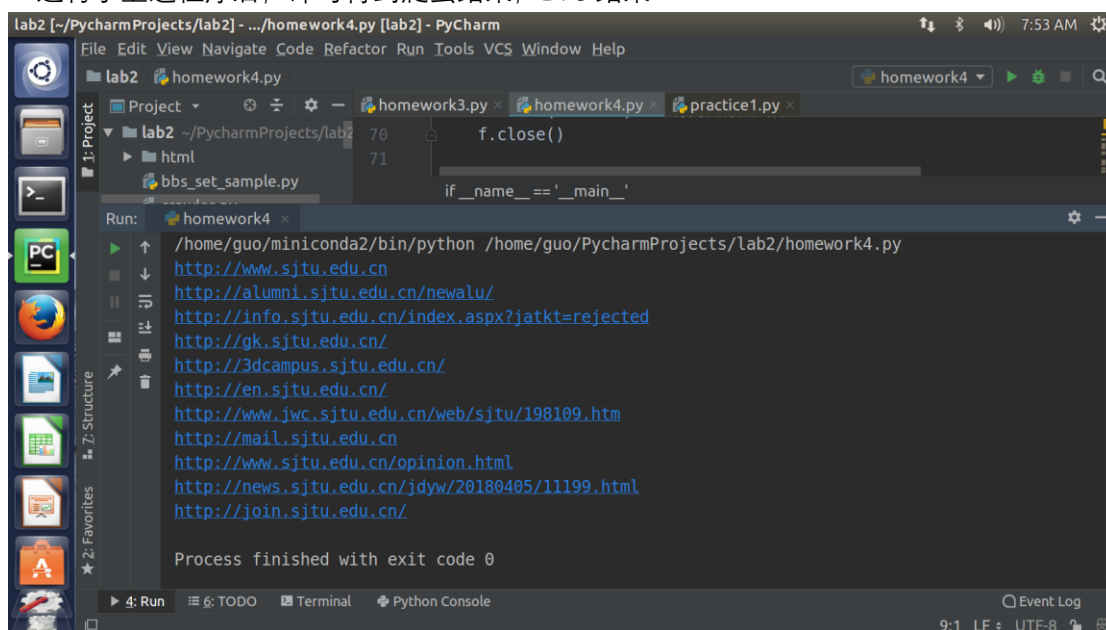
```
def get_page(page):
    try:
        content = urllib2.urlopen(page, timeout=10)
        content=content.read()
    except:
        return ''
    else:
        return content

def get_all_links(content, page):
    links = []
    # req = urllib2.Request(page, None, {'User-agent': 'guo'})
    # resp = urllib2.urlopen(req)
    soup = BeautifulSoup(content, "html.parser")
    urls=soup.findAll('a',{'href': re.compile('^http//')})
    for url in urls:
        t=url.get('href','')
        tmp=t.encode('utf8')
        if tmp[0]=='/':
            t=urlparse.urljoin(page,t)
        links.append(t)
    return links
```

在主函数里运行添加技术变量 count 即可控制爬虫数目，较容易。

4.2 实验结果

运行了上述程序后，即可得到爬虫结果，DFS 结果：

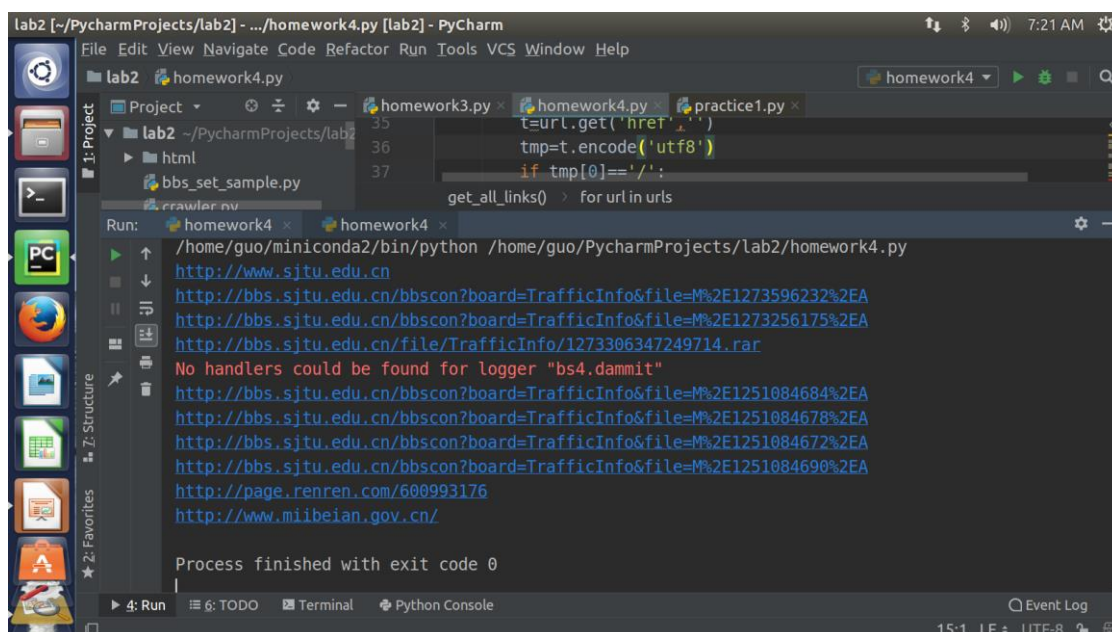


The screenshot shows the PyCharm IDE with the 'Run' console displaying the output of a DFS crawler. The output lists several URLs found on sjtu.edu.cn, including the homepage, alumni page, info page, and various departmental pages. The process finished with exit code 0.

```
Run: homework4 x
/home/guo/miniconda2/bin/python /home/guo/PycharmProjects/lab2/homework4.py
http://www.sjtu.edu.cn
http://alumni.sjtu.edu.cn/newalu/
http://info.sjtu.edu.cn/index.aspx?jakt=rejected
http://gk.sjtu.edu.cn/
http://3dcampus.sjtu.edu.cn/
http://en.sjtu.edu.cn/
http://www.jwc.sjtu.edu.cn/web/sjtu/198109.htm
http://mail.sjtu.edu.cn
http://www.sjtu.edu.cn/opinion.html
http://news.sjtu.edu.cn/jdyw/20180405/11199.html
http://join.sjtu.edu.cn/

Process finished with exit code 0
```

BFS 结果：



The screenshot shows the PyCharm IDE with the 'Run' console displaying the output of a BFS crawler. The output lists a larger number of URLs, including various bbscon links and external sites like renren.com and miibeian.gov.cn. A warning message about a logger is also visible. The process finished with exit code 0.

```
Run: homework4 x
/home/guo/miniconda2/bin/python /home/guo/PycharmProjects/lab2/homework4.py
http://www.sjtu.edu.cn
http://bbs.sjtu.edu.cn/bbscon?board=TrafficInfo&file=M%2E1273596232%2EA
http://bbs.sjtu.edu.cn/bbscon?board=TrafficInfo&file=M%2E1273256175%2EA
http://bbs.sjtu.edu.cn/file/TrafficInfo/1273306347249714.rar
No handlers could be found for logger "bs4.dammit"
http://bbs.sjtu.edu.cn/bbscon?board=TrafficInfo&file=M%2E1251084684%2EA
http://bbs.sjtu.edu.cn/bbscon?board=TrafficInfo&file=M%2E1251084678%2EA
http://bbs.sjtu.edu.cn/bbscon?board=TrafficInfo&file=M%2E1251084672%2EA
http://bbs.sjtu.edu.cn/bbscon?board=TrafficInfo&file=M%2E1251084690%2EA
http://page.renren.com/600993176
http://www.miibeian.gov.cn/

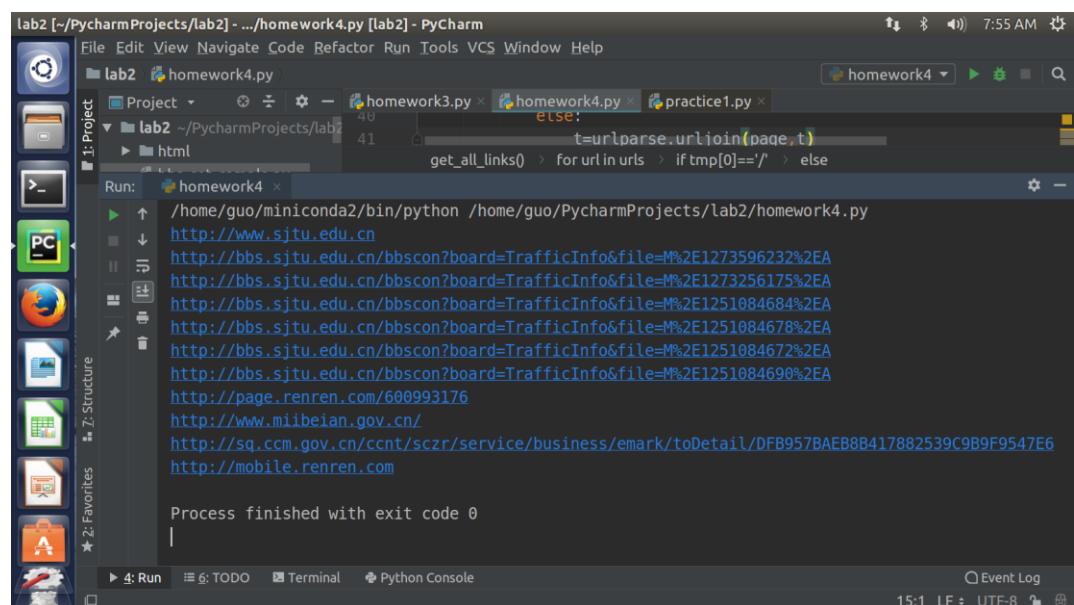
Process finished with exit code 0
```

NOTE：本实验以 <http://www.sjtu.edu.cn> 为 seed。需要对论坛等链接的相对地址使用 urljoin 得到绝对地址。

NOTE：DFS 搜索没有任何问题，在 BFS 搜索时，结果如图报错了一行红字，经过查询资料得知，因为如图爬取到的上一个链接是以.rar 结尾的下载链接，因此

内容过多无法完整存放在内存内，因此数据过大报错。我做了一些优化，在网页爬取之前判断是否为下载链接，如果是下载链接即跳过此爬取页面，优化后结果如下图。

```
def get_all_links(content, page):
    links = []
    # req = urllib2.Request(page, None, {'User-agent': 'guo'})
    # resp = urllib2.urlopen(req)
    soup = BeautifulSoup(content, "html.parser")
    urls=soup.findAll('a',{'href': re.compile('^http|^/')})
    for url in urls:
        t=url.get('href','')
        tmp=t.encode('utf8')
        if tmp[0]=='/':
            if tmp[-3:]=='rar':
                continue
            else:
                t=urlparse.urljoin(page,t)
        links.append(t)
    return links
```



(5) 练习 5

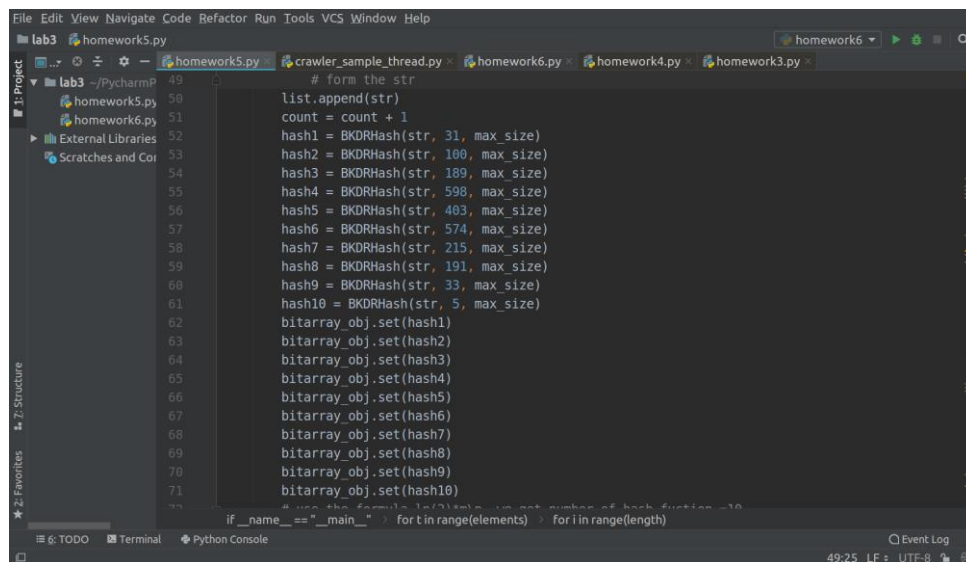
5.1 实验步骤

练习 5 需要实现一个自己的 bloomfilter 以及统计正确率。首先，我使用了 Bitarray 类记录哈希函数的结果，我使用了 10 个哈希函数，通过给同一个哈希函数 10 个不同的 seed 实现。如下图，list 作为一个记录器，所有存入的 url 都存入在 list 中。存入的数据我采用随机数的机制，随机一个 url 的长度，在随机一个相应长度对应的 url，通过 10 个哈希函数（10 个 seed）运算在 bloomfilter 内存入即可。我开启了一个 10000 长度的 bloomfilter，并向其添加了 500 个 url 进行测试。


```

if __name__ == "__main__":
    max_size = 10000
    # the total table /max_size is 10000
    bitarray_obj = Bitarray(max_size)
    elements = 500
    # we add 500 elements
    list = []
    for t in range(elements):
        length = random.randint(1, 100)
        # the length of str is from 1 to 100 (random)
        count = 0
        str = ''
        for i in range(length):
            str = str + chr(random.randint(32, 127))
            # form the str
        list.append(str)
        count = count + 1

```



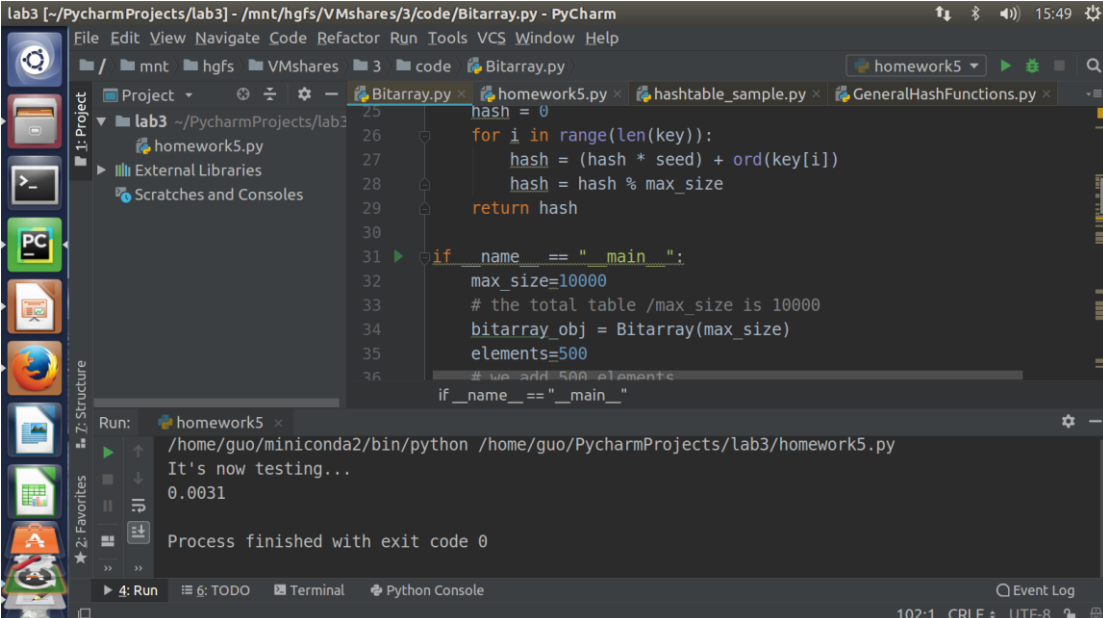
在测试环节，使用同样的方法，随机测试 url 的长度，然后再随机生成测试的字符串 str，如下图，我随机生成了 10000 个测试 url，只要其哈希函数的值符合原先的结果但其不在原 list 里即记错误一次。

```

print "It's now testing..."
test_num = 10000
false_count = 0
for i in range(test_num):
    length = random.randint(1, 100)
    # the length of str is from 1 to 100 (random)
    str = ''
    for i in range(length):
        str = str + chr(random.randint(32, 127))
        # form the testing str
    test_hash1 = BKDRHash(str, 31, max_size)
    test_hash2 = BKDRHash(str, 100, max_size)
    test_hash3 = BKDRHash(str, 189, max_size)
    test_hash4 = BKDRHash(str, 598, max_size)
    test_hash5 = BKDRHash(str, 403, max_size)
    test_hash6 = BKDRHash(str, 574, max_size)
    test_hash7 = BKDRHash(str, 215, max_size)
    test_hash8 = BKDRHash(str, 191, max_size)
    test_hash9 = BKDRHash(str, 33, max_size)
    test_hash10 = BKDRHash(str, 5, max_size)

```

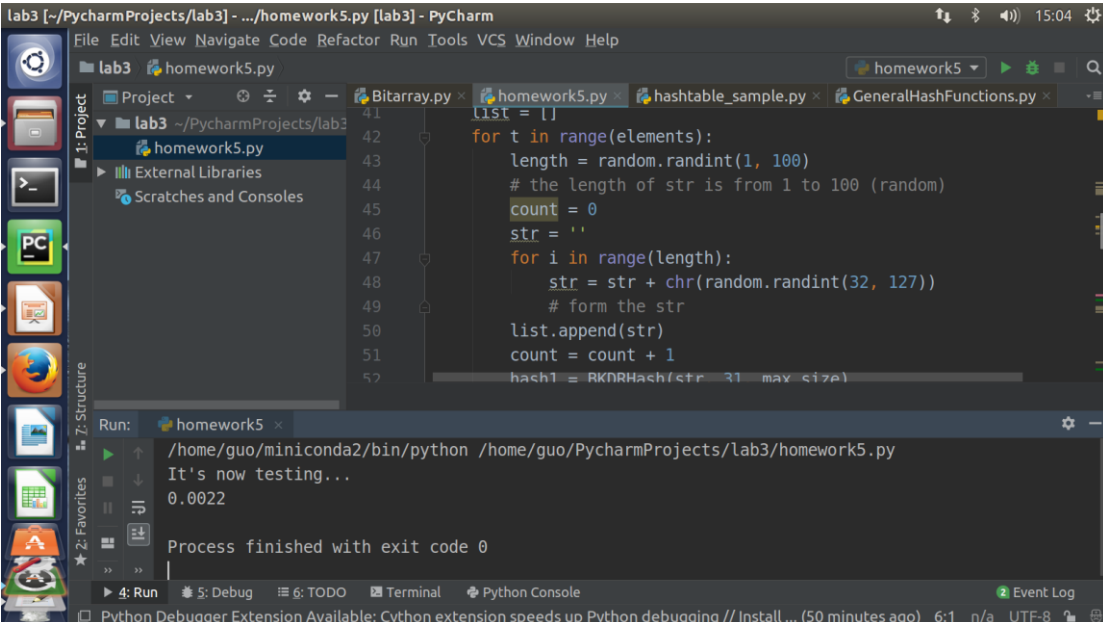
5.2 实验结果



```
lab3 [~/PycharmProjects/lab3] - /mnt/hgfs/VMshares/3/code/Bitarray.py - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Project: lab3 ~/PycharmProjects/lab3
  Bitarray.py
  homework5.py
  hashtable_sample.py
  GeneralHashFunctions.py
25 hash = 0
26 for i in range(len(key)):
27     hash = (hash * seed) + ord(key[i])
28     hash = hash % max_size
29 return hash
30
31 if __name__ == "__main__":
32     max_size=10000
33     # the total table /max_size is 10000
34     bitarray_obj = Bitarray(max_size)
35     elements=500
36     # we add 500 elements
37     if __name__ == "__main__":
38         ...

Run: homework5 x
/home/guo/miniconda2/bin/python /home/guo/PycharmProjects/lab3/homework5.py
It's now testing...
0.0031

Process finished with exit code 0
```



```
lab3 [~/PycharmProjects/lab3] - .../homework5.py [lab3] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Project: lab3 ~/PycharmProjects/lab3
  Bitarray.py
  homework5.py
  hashtable_sample.py
  GeneralHashFunctions.py
41 list = []
42 for t in range(elements):
43     length = random.randint(1, 100)
44     # the length of str is from 1 to 100 (random)
45     count = 0
46     str = ''
47     for i in range(length):
48         str = str + chr(random.randint(32, 127))
49         # form the str
50     list.append(str)
51     count = count + 1
52     hash1 = BKDRHash(str, 31, max_size)
```

经过测试，如上图所示，错误率在 0.22%、0.31%左右波动，可以达到 < 0.4%的稳定错误率，正确率稳定高于 99.6%，效率极高，速度也非常快。

(6) 练习 6

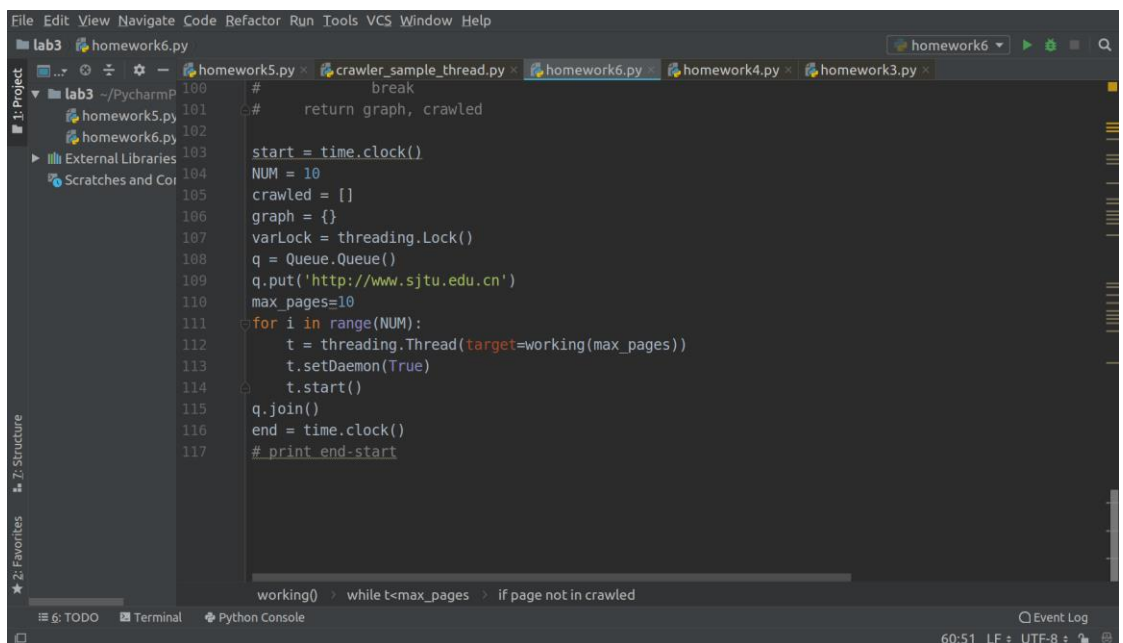
6.1 实验步骤

将实验 2 中的 crawler 改为并行实现。需要用到队列 Queue，每次弹出一个 url 给线程工作，并且使用 threading 库开启多线程。同时要注意，此时 crawled 等是全局变量，要使用 Varlock 保护线程不会同时对这些参数进行修改。


```
def get_page(page):
    print 'downloading page %s' % page
    # time.sleep(0.5)
    try:
        content = urllib2.urlopen(page, timeout=10)
        content=content.read()
    except:
        return ''
    else:
        return content
```

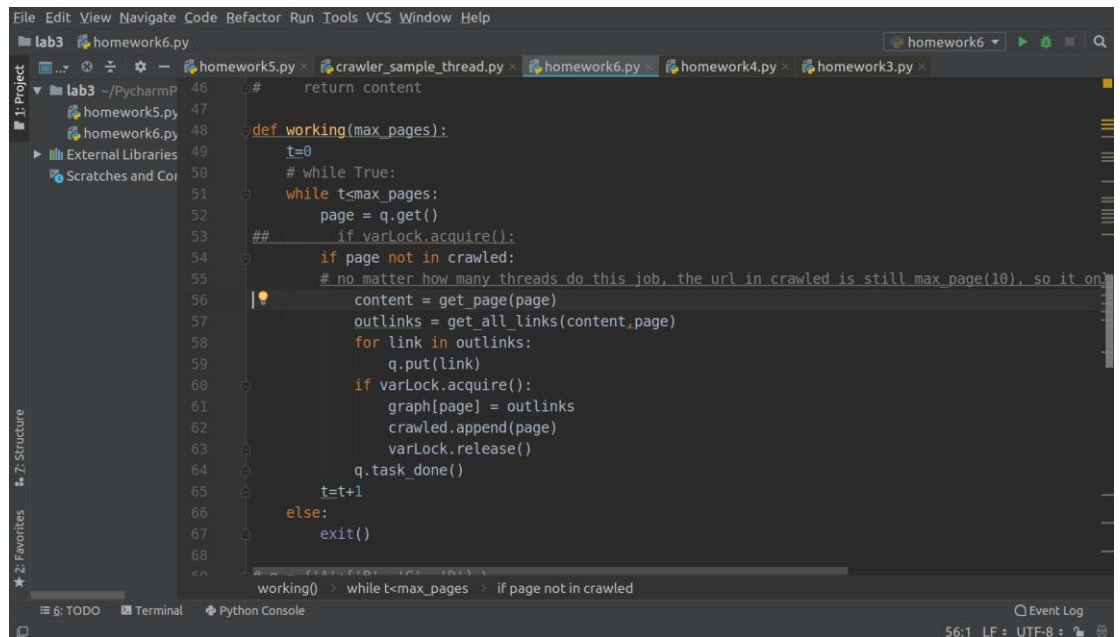
```
def get_all_links(content, page):
    links = []
    # req = urllib2.Request(page, None, {'User-agent': 'guo'})
    # resp = urllib2.urlopen(req)
    soup = BeautifulSoup(content, "html.parser")
    urls=soup.findAll('a',{'href': re.compile('^http|^/')})
    for url in urls:
        t=url.get('href')
        tmp=t.encode('utf8')
        if tmp[0]=='/':
            if tmp[-3]=='rar':
                continue
            else:
                t=urlparse.urljoin(page,t)
        links.append(t)
    return links
```

对 get_page 以及 get_all_links 进行修改。



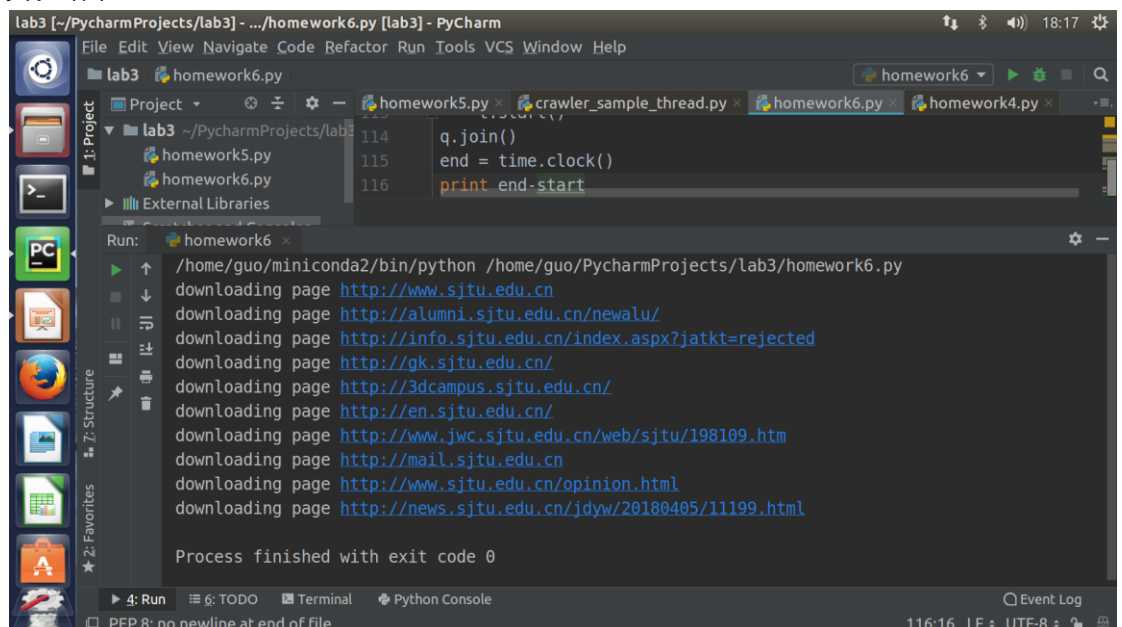
```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
lab3 homework6.py
homework5.py x crawler_sample_thread.py x homework6.py x homework4.py x homework3.py x
lab3 ~/PycharmP
homework5.py
homework6.py
External Libraries
Scratches and Coi
100 # break
101 # return graph, crawled
102
103 start = time.clock()
104 NUM = 10
105 crawled = []
106 graph = {}
107 varLock = threading.Lock()
108 q = Queue.Queue()
109 q.put('http://www.sjtu.edu.cn')
110 max_pages=10
111 for i in range(NUM):
112     t = threading.Thread(target=working(max_pages))
113     t.setDaemon(True)
114     t.start()
115 q.join()
116 end = time.clock()
117 # print end-start
working() > while t<max_pages > if page not in crawled
```

对 working 添加参数 max_pages。



NOTE :我对 working 函数传入了参数 max_pages，作为计数总数控制爬虫的个数，本实验我才用了 10 作为 max_pages。在 working 其中定义了计数变量 t，但按理说 t 在每一个线程操作时都会被重置于 0，n 个线程操作应该输出 10*n 个结果，但是这种写法仍是正确的，输出就是 10 个 url。因为计数变量 t 控制了得到的 page 数，每 get 一次，输出一次，因此不管多少线程对其操作，结果仍为每次 get.page 这 10 次操作，故程序正确。

6.2 实验结果



NOTE :如上图，上一个 NOTE 的论证正确，计数控制完美完成了目标没有错误。

3. 实验总结

此次试验是我第二次进行爬虫实验，对爬虫的认识逐渐加深。同时使用了哈希检索等优化方法，加快了爬虫的速率与效率，对算法优化的理解也渐渐深刻了。

此次实验还让我认识了 threading、urllib2 等极为实用的 python 库，同时给我的最大印象就是，在爬虫与 network 操作之前一定要搞清楚网页的信息结构与信息的提交方式，如 post\get 等，定向地寻找需要的信息。同时也让我体会到了两种搜索方式 BFS 以及 DFS 的优劣。

最后的实验也让我理解了并行对运算速度的巨大提升，并程序对效率提升巨大，是对算法的极大优化。

在编程过程中也遇到了很多困难和 Bug，还有很多思考了很久得出的结论与解释，都在上述“2.实验步骤”中，用加粗的 **NOTE** 加以了说明。

最后希望在之后的课程学习中对爬虫和相应知识有着更深的了解，继续努力！

o(*≥▽≤)ツ！

517030910374

郭嘉宋