实验 2-Crawler

一、实验概览

通过对 HTTP 协议的定义及在 python 中的模拟访问、爬虫的概念及抓取方式（优先性、礼貌性）爬取一定量的网页（练习二）之后利用 BloomFilter 来加速查重过程（练习一），最后尝试对练习二的方式进行并行化的处理，加快运行效率（练习三）。

二、实验环境

1. 个人笔记本电脑
2. 操作系统：windows10 专业版
3. 使用软件：Visual Studio Code；Docker Desktop

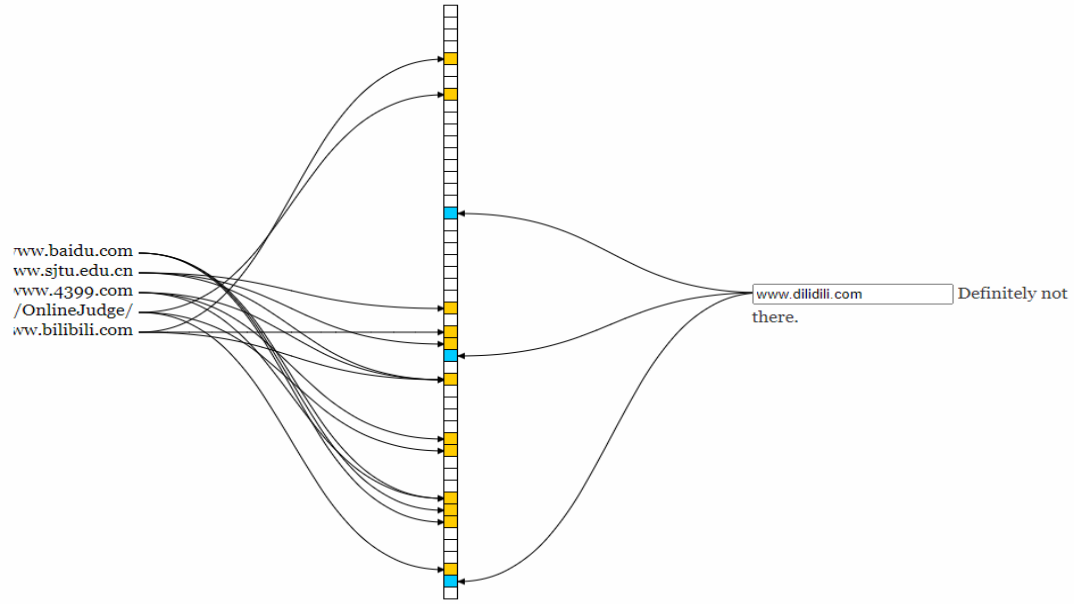
三、实验过程

练习一：实现一个简单的 Bloom Filter。并且设计一个实验统计你的 Bloom Filter 的错误率(false positive rate)。提示：可以用函数实现（例如 hashtable 里，用函数操作 table 的做法），也可以用类实现（例如 Bitarray.py 的实现，可以修改 Bitarray.py 完成 Bloom Filter）。

【解】

为了方便解决这道问题，我将其分为两个部分：①实现 Bloom Filter；②设计实验检验其容错率。并且从两个部分分别考虑，从而简化整个练习。

1. 实现 Bloom Filter



我

们可以用上图来简单表示 Bloom Filter 的实现原理[1]。通过这张图，我们可以发现，通过 Bloom Filter 的操作，我们利用多个函数将网址映射到二进制数组的不同位置上去，一个 URL 如果全部都在这个数组内，说明这个 URL 很有可能是已经重复的：

```
cnt = 0
for i in words:
    tmp, flag = 0, 0
    for j in range(0, 8):
        # use the mod calculation to avoid the set being overflowed
        tmp = Hash(i) % scale
        # to check whether this url:tmp is in this array
        if bitarray_obj.get( tmp ) :
            flag += 1
        bitarray_obj.set( tmp )
    # if it isn't in it, record it.
    if flag != 8:
        cnt += 1
```

通过这样的方式，我们可以构造出一个很好的 Bloom Filter 模型。当然，Bloom Filter 的二进制数组容量大小也是至关重要的，根据有关资料[2]，对于给定的 m、n，当 k = ln(2)* m/n 时出错的概率是最小的：

```
m = int(20*try_scale)
bitarray_obj = Bitarray(m+5)
```

但是这里仍缺少 Bloom Filter 中最重要的 Hash 函数的实现。这里我通过对 GeneralHashFunctions.py 的参阅，我选取了 BKDRHash 的函数并将其稍微变形以适应原本的程序：

```
def BKDRHash(key, j):
    # use "list" to make it easy when calling the function BKDRHash
    # j is the j_th base collect to calculate the hash value
    base = [31, 131, 1313, 13131, 131313, 1313131, 13131313, 131313131]
```

这样便可以方便我对于 Bloom Filter 模型的调用，进而成功实现 Bloom Filter。

2. 设计实验检验其容错率

在入手这一块之前，我需要确定大致的操作模型，思考后认为应该是：

①生成测试数据（大量的链接）；

②用 Bloom Filter 和一般方法来进行填充放入；

③检验容错率。

①首先，这里我利用 random 库进行随机单词的生成（此处利用单词没有利用 URL，但

[1] http://www.jasondavies.com/bloomfilter/ Bloom Filters

[2] http://pages.cs.wisc.edu/~cao/papers/summary-cache/node8.html BloomFilters- the math

是本质上是一样的），确定常见单词的长度为 1-12，故可以得到如下的单词生成代码：

```
    f = open("random.txt","w")
    while(n):
        l = random.randint(1,12)
        word=""
        while(l):
            # choose a random word from a to z, which is 1 to 26 in the rank
            tmp=random.randint(1,26)
            word+=chr(tmp+96)
            l-=1
        f.write(word+" ")
        n -= 1
    f.close()
```

②此处利用的一般方法就是 python 的 **in** 与 **not in** 的关键字（利用 list 或者 dict 等存储结构都可以实现这样的操作）：

```
dic = {}
for i in words:
    if i not in dic:
        dic[i] = 1
std = len(dic)
```

③只需要将 Bloom Filter 得到的“认为不同的 URL”个数与一般方法得到的“不同的 URL”的个数相比即可得到二者的容错率。

```
false_rate.append(1.0-cnt*1.0/std)
```

最后，为了保证实验的公平性，程序将进行多次运行并取平均值的方式来获得结果，经实验的尝试，我发现，当数据规模比较大的时候，利用 **Bloom Filter** 模型得到的容错率还是比较高的（失误率很低），具体程序实现在请见 Bitarray.py、randomtext.py 与 sumup.py。

练习二：实现一个简单的网页爬虫（修改 **crawler.py**）。

【解】

首先进行 get_all_links(content, page)的修改，此处的关键在于爬取下来的所有 a 标签 href 属性的链接均需要满足正常链接的方式（以 http 开头或者以/开头的链接），利用 BeautifulSoup 自带的 findAll 函数以及正则表达式可以很方便地解决这个问题：

```
def get_all_links(content, page):
    links = []
    soup = BeautifulSoup(content)
    for i in soup.findAll('a',{"href" : re.compile("^http|^/")}):
        t = i.get('href','')
        #将相对链接转化为绝对连接（如果是 http 开头则不会改变）
        t = urllib.parse.urljoin(page, t)
```

```
        if t not in links:
            links.append(t)
    return links
```

其次进行 get_page(page)的修改，此处的关键在于对异常的处理，当爬取到存在异常的网页时，需要进行一定的处理以规避程序崩溃的尴尬情形（例如抛出异常、返回空网页）：

```
def get_page(page):
    n = 10
    #设定最多进行 10 次重连尝试
    while(n):
        try:
            flag = False
            # 给予一定的访问时间缓冲 timeout
            content = urllib.request.urlopen(page,timeout=2000).read()
            return content
        except:
            # 告诉用户具体哪个网页在访问时出了问题
            print("This is the {0}th try for the website {1}".format(n,page))
            n-=1
    return None
```

由于返回的是 None 类型，故此时需要在主程序进行一定的判断来说明该网页访问不了并跳过，否则将会对主程序的判断产生影响而致崩溃：

```
# 获取网页内容
content = get_page(page)
# 发现获取的是空的，即获取失败
if not content:
    # 此处设立了一个无法访问网页的记录列表可以供使用者参考
    error_url.append(page)
    continue
```

最后，在爬取到最大量(maxsize)的网页的时候需要及时退出以防程序陷入死循环：

```
# 爬取完毕，已经爬取的网页数+1
count += 1
if count >= max_page:
    return crawled
```

进而，我们的练习二就这样完成了。

练习三：实现一个并行的爬虫。

将练习 2 中的 **crawler.py** 改为并行化实现(partD/**crawler_multi_thread.py**)。

【解】

并行化的操作实际上就是利用 python 的 Queue 类来进行多线程的操作，Queue 内的第一个值为待爬取网页，之后将每次网页中爬取的未被函数调用的链接存入队列中进行重复操作：

```
page = q.get()
if page not in crawled:
    content = get_page(page)
            #网页如果访问无效则本次任务结束，继续进行之后的任务
    if not content:
        q.task_done()
        continue
    outlinks = get_all_links(content, page)
    add_page_to_folder(page, content)
    for link in outlinks:
        q.put(link)
    graph[page] = outlinks
    crawled.append(page)
    q.task_done()
```

注意到，实际上有的网页是始终存在大量其他网页的“友链”，那么如果我们爬取的对象如果比较特殊，我们想要爬尽网页内的所有链接并继续遍历所有链接，所耗费的时间代价是非常巨大的，甚至无法在有限时间内爬取完毕，因此，设立计数器及时退出程序是非常有必要的，并且在多线程的情况下，如果要调用同一个变量进行操作，最好利用 varLock 来保护这个变量 count 以防止出现异常：

```
global count
varLock.acquire()
count += 1
if(count>=2000):
    varLock.release()
    q.task_done()
    return
```

但是这么做的情况是我的程序陷入了死循环，这成为了我的一个思考（在**四、实验问题**中回答）。

对于这个实验中的 get_page 函数，为了保证我对这个实验程序运行情况的实时了解，我通过不断输出相应的文本来说明读取网页是否成功：

```
def get_page(page):
    n = 10
    while(n):
        try:
            print('downloading page %s' % page)
            content = urllib.request.urlopen(page,timeout=50).read()
            return content
        except:
            print("This is the {0}th try for the website {1}".format(n,page))
            n-=1
    return None
```

进而，我们的练习三得到了解决。

四、实验问题及解决

问题 1：在练习三中的 get_all_links 函数中，我利用 BeautifulSoup 进行网页链接的处理，这样的处理速度比较慢，能不能利用 str 类型转换成更快的链接处理方式呢？

【解】

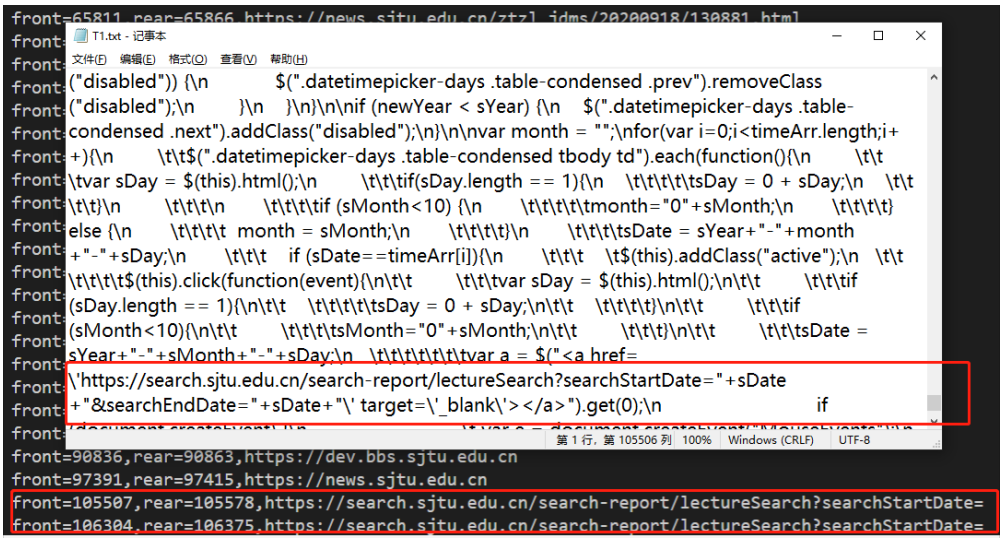
由于在 BeautifulSoup 的处理过程中处理速度是比较慢的，我的第一个想法是将其转换成字符串 str 的形式，然后利用字符串的 find 函数查找一些符合特征的链接然后爬取下来：

```
content = urllib.request.urlopen("https://www.sjtu.edu.cn/",timeout=500).read()
soup = str(content)
while(soup.find("href",front)!=-1):
    #爬取的有 http 开头或者 https 开头的链接
    front = soup.find("http:/",front)
    #front = soup.find("https:/",front)
    #这些链接基本是以单引号或者上引号结尾的
    rear1 = soup.find("\'",front)
    rear2 = soup.find('\"',front)
    #因为无法判断，所以选取二者的最小
    rear = min(rear1, rear2)
    #如果链接不为空则存入爬取链接中（https 则为 8）
    if(rear-front>7):
        links.append(soup[front:rear])
        print("front={0},rear={1},{2}".format(front,rear,soup[front:rear]))
    #继续爬取
    front = rear
```

但是，理想很丰满，现实很骨感，在实现的过程中我遇到了几个问题：

①爬取不完整：存在部分网页的链接是包含单、双引号的，从而导致链接爬取不全。

②相对路径缺失：不能够以 // 来作为一个 find 的查找判准，怎么查找呢？



通过对正则表达式提取网页链接的学习[3]，可以发现，我们要提取的始终是 a 标签下 href 属性内的链接，所以只需要匹配并取出所包含的内容即可（这里利用小括号即可提出）。

所以对于所有的绝对链接（指以 http 或者 https 开头的链接），我们均可以利用正则表达式：<a href="([a-zA-z]+://[^\s]*)"来进行抓取链接内的东西，对于所有的相对链接，只需要匹配出所有的<a href="(/[^\s]*)"后再进行补全即可：

```
1. url = "https://www.sjtu.edu.cn/"
2. soup = urllib.request.urlopen(url).read()
3. tag1 = re.findall(r'<a href="([a-zA-z]+://[^\s]*)"', str(soup))
4. tag2 = re.findall(r'<a href="(/[^\s]*)"', str(soup))
5. print(tag1)
6. tag2 = [url+i for i in tag2]
7. print(tag2)
```

这样运行得到的结果为：

```
['https://www.sjtu.edu.cn/', 'https://www.sjtu.edu.cn/', 'https://ddh11.sjtu.edu.cn/', 'https://news.sjtu.edu.cn/jdyw/20200928/131635.html',
 'https://news.sjtu.edu.cn/jdyw/20200929/131725.html', 'https://mp.weixin.qq.com/s/N1HkKmGZGU5TCW-AXvh8sQ', 'https://news.sjtu.edu.cn/jdyw/2
0201004/131951.html', 'https://news.sjtu.edu.cn/ztzl_djfcxl/20200930/131761.html', 'https://news.sjtu.edu.cn/jdyw/20200930/131773.html', 'ht
tps://news.sjtu.edu.cn/ztzl_dyfc/20200930/131755.html', 'https://news.sjtu.edu.cn/jdyw/20200930/131771.html', 'https://news.sjtu.edu.cn/jdyw
/20200930/131871.html', 'https://news.sjtu.edu.cn/jdyw/20200930/131817.html', 'https://news.sjtu.edu.cn/jdyw/20201001/131911.html', 'https:/
/news.sjtu.edu.cn/jdyw/20200930/131739.html', 'https://news.sjtu.edu.cn/jdyw/20200930/131875.html', 'https://news.sjtu.edu.cn/jdyw/20200930/
131877.html', 'https://news.sjtu.edu.cn/jdyw/20200930/131745.html', 'https://news.sjtu.edu.cn/jdyw/20200930/131865.html', 'https://news.sjtu
.edu.cn/jdyw/20200930/131785.html', 'https://news.sjtu.edu.cn/index.html', 'https://www.sjtu.edu.cn/tg/index.html', 'https://news.sjtu.edu.c
n/mtjj/20200929/131667.html', 'https://news.sjtu.edu.cn/mtjj/20200927/131549.html', 'https://news.sjtu.edu.cn/mtjj/20200928/131611.html', 'h
ttps://news.sjtu.edu.cn/mtjj/20200927/131551.html', 'https://news.sjtu.edu.cn/mtjj/index.html', 'https://news.sjtu.edu.cn/tsfx/index.html',
'https://news.sjtu.edu.cn/jdzh/20200927/131463.html', 'https://news.sjtu.edu.cn/jdzh/20200928/131621.html', 'https://news.sjtu.edu.cn/jdzh/2
0200928/131625.html', 'https://news.sjtu.edu.cn/jdzh/20200928/131623.html', 'https://news.sjtu.edu.cn/jdzh/20200923/131259.html', 'https://n
ews.sjtu.edu.cn/xsjz/index.html', 'https://news.sjtu.edu.cn/ztzl/index.html', 'https://news.sjtu.edu.cn/ztzl_jdms/20200929/131687.html', 'ht
tps://news.sjtu.edu.cn/ztzl_fwzx/20200925/131383.html', 'https://news.sjtu.edu.cn/ztzl_jdms/20200924/131327.html', 'https://news.sjtu.edu.cn
/ztzl_fwzx/20200928/131571.html', 'https://news.sjtu.edu.cn/ztzl_jdms/20200918/130881.html', 'https://news.sjtu.edu.cn/ztzl_fwzx/20200921/13
1081.html', 'https://news.sjtu.edu.cn/ztzl/index.html', 'https://news.sjtu.edu.cn/hlxy/index.html', 'https://news.sjtu.edu.cn/hlxy/20200925/
131401.html', 'https://news.sjtu.edu.cn/hlxy/20200915/130663.html', 'https://news.sjtu.edu.cn/hlxy/20200820/129423.html', 'https://news.sjtu
.edu.cn/hlxy/20190620/105575.html', 'https://news.sjtu.edu.cn/hlxy/20200915/130665.html', 'https://news.sjtu.edu.cn/hlxy/20200915/130661.htm
l', 'https://news.sjtu.edu.cn/hlxy/20200102/118851.html', 'https://news.sjtu.edu.cn/hlxy/20200824/129529.html', 'http://my.sjtu.edu.cn/', 'h
ttps://houqin.sjtu.edu.cn/service.php?cid=26', 'http://www.lib.sjtu.edu.cn/', 'http://mail.sjtu.edu.cn', 'http://electsys.sjtu.edu.cn', 'htt
p://vi.sjtu.edu.cn/', 'https://www.sjtu.edu.cn/opinion/index.html', 'http://xygl.sjtu.edu.cn', 'http://bwc.sjtu.edu.cn', 'http://info.sjtu.e
du.cn/index.aspx?jatkt=rejected', 'https://net.sjtu.edu.cn/', 'http://gk.sjtu.edu.cn/', 'https://oc.sjtu.edu.cn', 'https://dev.bbs.sjtu.edu.
cn', 'http://lc.sjtu.edu.cn', 'http://postd.sjtu.edu.cn']
['https://www.sjtu.edu.cn//tg/20200922/131229.html', 'https://www.sjtu.edu.cn//tg/20200929/131659.html', 'https://www.sjtu.edu.cn//tg/202009
22/131189.html', 'https://www.sjtu.edu.cn//tg/20200929/131701.html', 'https://www.sjtu.edu.cn//tg/20200930/131797.html']
```

可以较为成功地爬取出所有包含的网页（其中 tag1 为绝对链接，tag2 为相对链接）。

将其替换入我们的多进程爬取程序中，然后判断其的优化程度，此处的判断是以爬取规模为 300，在线线程为 8 条的爬取情况：

old300	new300
15.83637	12.76832
17.57993	14.01853
15.73175	12.27501
16.38268	13.02062

可以发现，这个方法整体在规模较大的情况下比较优。

代码呈现于 crawler_multi_thread.py 内（与练习三一并提交）。

问题 2：在练习三的实现过程中，如果在主程序的语句中加入了 q.join()则程序无法停止，陷入无响应的状态，此时应该怎么处理以保证程序正常运行？

【解】

刚开始我注意到，在主程序中线程是以守护线程的形式出现的：

```
1. for i in range(NUM):
2.     t = threading.Thread(target=working)
```

[3] https://www.runoob.com/regexp/regexp-syntax.html 正则表达式语法学习

```
3.      t.setDaemon(True)
4.      t.start()
```

考虑到守护线程的特殊性质：当主程序结束的时候守护线程也将同时结束。我利用主程序的循环判断的方式来作为结束条件，其中 count 为我们已经操作的链接数：

```
1.  while(count<100):
2.      time.sleep(1)
3.      pass
```

并且将 q.join()注释掉，因为 q.join()结束当且仅当所有线程的任务结束，但是我们在操作中即便利用了 return 以及那个任务的 q.task_done()，也仅仅是体现在那个任务中，与其相关的其他线程也仍旧在继续工作，从而导致 q.join()语句始终挂起。

但是，这样处理的问题在于，我虽然能够及时退出整个程序，但是在结束的瞬间我的 q 是仍旧存在大量积存任务未清空的，而这样的想法显然对于我们的实际编程应用是不利的（可能因此而导致异常退出或者程序崩溃等等的情况），因此我需要去思考有没有更优的方法。

然后我想到了，如果在 count 一满足条件就清空 q 并退出是不是就可以结束了？答案是可以的。这里我利用 while 语句将目前的所有线程中止，然后及时退出函数，从而满足要求：

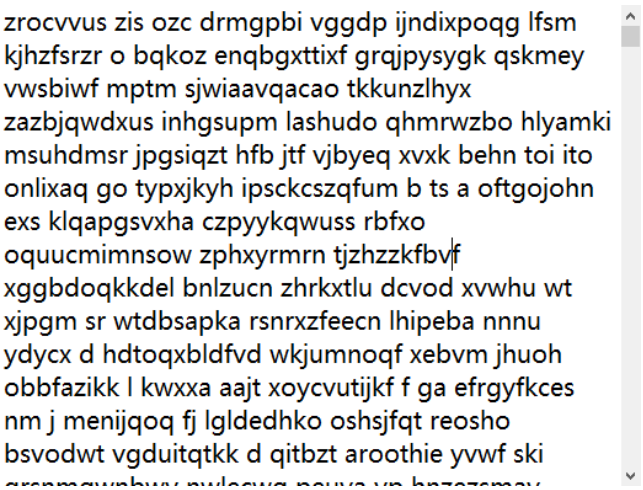
```
1.  if(count>=2000):
2.      varLock.release()
3.      while 1:
4.          try:
5.              q.task_done()
6.          except:
7.              break
8.      return
```

利用 try/except 语句可以完美归并处理过程中出现的 q 空仍旧清空 q 的错误问题。

五、实验结果

练习一（节选）：

（random.txt:生成大量的单词）：



实验精度（错误率）以及附上的相应规模：

```
Now it's 10 to try, and the scale is 10000, the rate is 0.00011261261261263922
Now it's 9 to try, and the scale is 20000, the rate is 0.0
Now it's 8 to try, and the scale is 40000, the rate is 0.0
Now it's 7 to try, and the scale is 80000, the rate is 0.0
Now it's 6 to try, and the scale is 160000, the rate is 0.0
Now it's 5 to try, and the scale is 320000, the rate is 0.0
Now it's 4 to try, and the scale is 640000, the rate is 0.0
Now it's 3 to try, and the scale is 1280000, the rate is 1.0351270359754139e-06
Now it's 2 to try, and the scale is 2560000, the rate is 2.1111064784085087e-06
Now it's 1 to try, and the scale is 5120000, the rate is 1.3509135823319696e-06
1.171097597093551e-05
```

```
Now it's 10 to try, and the scale is 5000000, the rate is 3.593913238009172e-06
Now it's 9 to try, and the scale is 5000000, the rate is 1.934717651419504e-06
Now it's 8 to try, and the scale is 5000000, the rate is 3.596092430613318e-06
Now it's 7 to try, and the scale is 5000000, the rate is 4.424612348619128e-06
Now it's 6 to try, and the scale is 5000000, the rate is 3.3182546312904293e-06
Now it's 5 to try, and the scale is 5000000, the rate is 5.529169131768263e-06
Now it's 4 to try, and the scale is 5000000, the rate is 3.31760878924392e-06
Now it's 3 to try, and the scale is 5000000, the rate is 4.6999444853756955e-06
Now it's 2 to try, and the scale is 5000000, the rate is 3.872487827294435e-06
Now it's 1 to try, and the scale is 5000000, the rate is 4.423874656156457e-06
3.871067518979032e-06
```

练习二（节选）：

```
This is the 2th try for the website http://xueshu.cnki.net/
This is the 1th try for the website http://xueshu.cnki.net/
http://ad.cnki.net
http://www.cnki.net/
http://piccache.cnki.net/index/images/gb/170001.jpg
Some characters could not be decoded, and were replaced with REPLACEMENT CHARACTER.
http://piccache.cnki.net/index/images/gb/271.jpg
Some characters could not be decoded, and were replaced with REPLACEMENT CHARACTER.
https://ss.knet.cn/verifyseal.dll?sn=e131111110100433641sll000000&a=1&pa=0.08881110103584755
This is the 10th try for the website http://www.knet.cn/kexin
This is the 9th try for the website http://www.knet.cn/kexin
```

练习三（节选）：

```
downloading page http://net.sjtu.edu.cn
downloading page http://vi.sjtu.edu.cn/index.php/articles/base/1
downloading page http://vi.sjtu.edu.cn/index.php/articles/base/2
downloading page http://vi.sjtu.edu.cn/index.php/articles/base/3
This is the 10th try for the website http://mail.sjtu.edu.cn/profile/#apply
downloading page http://vi.sjtu.edu.cn/index.php/articles/base/4
downloading page http://mail.sjtu.edu.cn/profile/#applydownloading page http://vi.sjtu.edu.cn/index.php/articles/base/5
This is the 7th try for the website https://pubs.acs.org/doi/abs/10.1021/acssynbio.0c00161
This is the 10th try for the website http://mail.sjtu.edu.cn/profile/#/find
```

六、实验体会

这是本课程的第二次实验作业，这次的难度与上次其实有着较大的差异，因而完成起来并没有那么容易。在实验过程中，我逐渐了解了 Hash 算法、多线程爬虫以及正则表达式的应用。最后，对于练习中遇到的问题，我能够及时在网上或其他地方寻求帮助，这对我学习态度的

端正有着很好的帮助！

当然，本次实验也存在不足之处，主要是由于刚开始学习这样的一个新方向，导致了我在一些细节上的处理可能还比较不尽人意，希望在接下来的几次实验中我能够越做越好！