计算机编程实验-Lab 2

实验报告

**12组**

王 韬 2017111287

王丹阳2017111411

* 实验目标

本次实验的内容是设计并开发一个网站页面分析系统，以满足在尽可能短的时间内，对16万个URL完成爬取，构建页面链接网络，并对其进行相应的分析，通过计算每个URL的pagerank值，给出排名最高的前10个页面。

本次实验的意义在于学习设计高性能的网络爬虫，以实现高速并发抓取，并理解矩阵的作用、利用矩阵来建模图。

通过学习这些抓取数据方法，和对其进行相应分析的方法，我们可以在以后实际的研究中进行相应的拓展应用，例如：垃圾邮件过滤、网页去重、基于图结构分析社交网络或科研合作网络或欺诈网络等复杂网络。

* 系统架构设计

本次试验总体上可以分为两大部分：爬虫、以及计算每个URL的pagerank值并进行排序，生成前十名。其中，爬虫部分又可以分为四个步骤：

1、建立socket通信接口，并向NGnix服务器端发送请求，以及接收服务器返回的.html文件并进行相应的解析；

2、将网页用bloom过滤器进行去重处理并将符合条件的网页加入队列，同时将其下属的链接与之的map关系存储到内存中，该链接关系是最初始查询到的层级之间的链接关系；

3、本组的爬虫类型是广度优先搜索，因此，当所有同一层级的URL均被添加到队列中后，队列才会对其进行处理，然后再找所有该层级URL的子链接，以完成将下一层级的URL加入队列的任务。以此类推，直至完成将所有层级的链接都加入到队列的遍历；

4、生成无重复的链接关系，为下一步计算PageRank值做好准备；

* 系统实现过程

本组的实现过程主要可以分为以下6个步骤，下面，报告将逐一详尽介绍各个步骤。

一、前期准备

本组首先对linux系统环境进行了熟悉和学习，但由于系统匹配的问题，本组决定还是先用标准化语句在windows系统中进行编程开发的工作以后，再将程序整体转移到Linux系统中进行编译和运行。

本组也完成了在电脑上搭建Nginx和网站环境的工作，完成了本次实验的前期准备。

二、开发可构造HTTP请求协议的客户端（抽取超链接和去重）

在尝试与服务器进行连接的过程中，本组采用了Socket编程。鉴于Socket是一组接口、是应用层与TCP/IP协议族通信的中间软件抽象层这样的本质，Socket的设计模式可以把复杂的TCP/IP协议族隐藏在Socket接口后面。那么对于用户来说，这组Socket接口即为全部，用户只需要让Socket去组织数据，以符合所指定的协议。

当设定好Socket后，Ngnix服务器端会先初始化Socket，然后与相应的端口绑定(bind)，因为本次实验中用到的是HTTP请求和HTTP响应，因此所对应的端口号是80端口，之后，服务器要对该端口号进行监听 (listen)，调用accept阻塞，等待客户端连接。本次实验所给定的NGnix服务器具有高并发数的特性，因而，它一次可以接收相当多数目的请求，效率较高。

接下来，就需要在客户端设定Socket连接的工作，在初始化一个Socket之后，连接服务器(connect)，如果连接成功，则此时客户端与服务器端的连接就建立完毕了。客户端可以向服务器端发送数据请求，服务器端则要进行接收请求的操作，并对该请求进行处理，然后再将响应的数据回发给客户端。客户端会读取相应数据，最后关闭连接，至此，一次交互完成。本组实验中，客户端所接收到的回复是一个.html文件。



本组设计客户端会在接收到.html文件后，对其中的“href”关键词进行字符串匹配识别，若发现了该关键词，则说明该URL对应的页面中还包含了下一级的URL链接。此处，本组应用了广度优先搜索的爬虫模式，因而会将全部第一层级的链接爬取完毕后，才会爬取所识别到的下一级链接，因此，在客户端发现了服务器对该URL回复的页面中仍包含下一层级URL链接时，它会先将二者的map关系存入内存中，为后续的操作提供准备。

在此，进一步讨论一下广度优先搜索策略。[广度优先搜索](https://baike.baidu.com/item/%E5%B9%BF%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2)是指：在抓取过程中，在完成当前层次的搜索后，才进行下一层次的搜索。

另外，本组是通过有限状态自动机的算法完成对“href”关键词进行字符串匹配的。其逻辑结构如下图所示：



由于每个层级所包含的下属链接可能会与该层级的其他链接重复，因此需要筛选等待被加入队列的链接，如果该链接已经在队列中了，则无需让该链接仍然存在在等待队列中，直接筛掉即可。这里，本组应用的是Bloom过滤器来执行这部分的去重工作。

Bloom过滤器的URL去重机制，是用于检索一个元素是否在一个集合中。其核心思想是：用准确率换取空间，因此它的空间效率和查询时间非常高。即：有n个元素集合S={x1, x2,...,xn}，一个包含m位的二进制位数组存储，K个相互独立的哈希函数映射到{1,...,m}的范围，S集合中的每个元素用k个hash函数映射到，{1,...,m}范围内，将相应的位置为1。

但基于这样的机制，Bloom过滤器可能会有一定的误识别率，它会将有些未出现的、但是其哈希映射值与已出现的链接的哈希映射值一致的链接也同样筛选掉，因此，它可以保证绝对没有重复的情况出现，但是不能保证结果是一定完全的。也就是说，虽然Bloom过滤器会控制整体的数目，保证不会出现重复的链接，但当多个链接的映射为同一个时，即使第二个还未出现，但它以后也都不会出现了。

本组通过相应公式的指示以及实际的实践，将m值设置为9，n的值为1000万，也就是说一共采用了9个哈希，1000万个bit，完成了对Bloom过滤器的设置。

三、加入libevent、非阻塞设计模式、以及I／O多路复用

接下来，在完成了去重的工作同时，这些符合要求的URL也已经被塞入队列中，等待解析。

为了尽可能地降低程序整体的时间复杂度，本组在多线程这部分运用了Libevent工具，将socket设置成了非阻塞模式，也在爬虫部分运用了I/O复用(multiplexing)的思想。这种利用非阻塞IO及select进行程序的串行执行的方法，会减少同步消耗，提高程序的执行速度。下面，本报告将对这三种提升程序整体运行速度的方法进行一一介绍：

1、 基于Libevent异步IO库的多线程并发抓取。

这个方法的核心思想是：一个基于事件触发的网络库。 这个库具有事件驱动(event-driven)、高性能、轻量级、专注于网络、跨平台（支持Windows、Linux、\*BSD和MacOs等多种操作系统）、支持多种I/O多路复用技术（如：epoll、poll、dev/poll、select和kqueue等）、支持I/O，定时器和信号等事件的特点。

Libevent已被广泛应用为底层的网络库，比如：memcached、Vomit、Nylon、Netchat等等。Libevent代码里有很多有用的设计技巧和基础数据结构，比如信息隐藏、函数指针、c语言的多态支持、链表和堆等。

它的运行原理在于：使用Libevent也是向Libevent框架注册相应的事件和回调函数，当这些事件发生时，Libevent会调用这些回调函数处理相应的事件(I/O读写、定时和信号)。本组将建立socket、发送HTTP Request、返回HTTP Response这三种动作均看成是事件（event），并将这些事件都放入event\_base中，由于base的设置模式为并发非阻塞的，因此在运行event\_base的同时，向base中不断添加event事件的操作仍然会持续进行，因而，整个程序的运行效率会得到大幅度的提升。

程序结束；

是

URL队列是否为空？

否

否

当前链接数<并发数？

通知event\_base开始处理所有的注册事件（调用callback函数以进行非阻塞的处理模式）；

是

创建建立socket的事件，并注册到event\_base中；

创建request请求，与socket连接，注册到event\_base中；

2、 非阻塞I/O模式

在该模式下，即使没有数据，socket也不会进入等待，而是立即返回特定错误。实现方法是使用ioctlsocket进行设置。这种方法提高了socket的运行速度。

3、 I/O复用

该方法是指：查询多个socket可读或可写是否准备好的状态，并发服务器管理多个客户连接IO、本地文件IO等，并要求为非阻塞Socket。由于在Windows系统下，Select是其经典的实现方式，但它的并发数有限制，最多为1024；在Linux系统下，基于内核通知的epoll是其的实现方式，这种多连接时、高性能的特点会提升整体的程序运行效率。本组也运用了回调函数，使得高并发的I/O服用可以同时处理多个socket。

四、保存URL和链接关系到文件

本组利用保存了所有超链的图结构和文件结构，并进行持久化存储到.txt文件中。在保存连接关系的过程中，本组认为，可以有以下两种方法：1、用find函数；2、用Bloom过滤器。

本组采用的是第二种方法，即：先建立各个链接之间的关系，然后用Bloom过滤器筛选出其中重复的关系对。

五、 利用矩阵迭代算法计算PageRank、Top10的重要网页

在PageRank部分，该算法是基于“从许多优质的网页链接过来的网页，必定还是优质网页”的回归关系，来判定所有网页的重要性。

其核心思想是：一个节点的“得票数”由所有链向它的节点的重要性来决定，到一个节点的边相当于对该节点投一票。一个节点的PageRank是由所有链向它的节点的重要性经过递归算法得到的。一个有较多链入的节点会有较高的等级，相反如果一个节点没有任何链入边，那么它没有等级。

例如：链向网页E的链接远远多于链向网页C的链接，但是网页C的重要性却大于网页E。这是因为因为网页C被网页B所链接，而网页B有很高的重要性。因此，一个网页的PageRank值取决于两个方面：该网页所关联的链接数目、以及与之关联的链接的PageRank值。

本组采用了基础的稀疏矩阵存储模式对通过计算所得到的矩阵进行了存储。

六、 将整体程序迁移到Linux系统环境下

本组的实现方法是先在windows上进行libevent的标 准实现，再通过去掉：

#include <WinSock2.h>

#include "queue.h"

WSADATA wsa\_data;

WSAStartup(MAKEWORD(2, 2), &wsa\_data);

以及添加：linux中网络编程相关的：

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <netinet/in.h>和strstr一些函数相关的：

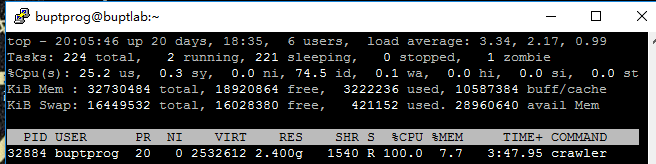
#include <string.h>

并更改Sleep为sleep，（Sleep中1000是1s，小sleep中1是一秒），

最终得到在linux上可编译的程序。

* 实验结果

本组程序爬虫部分在运行中所占用的内存约为3146.71M，主要占用内存的部分是大量的socket和最原始存入内存的链接关系。内存显示如下图所示：



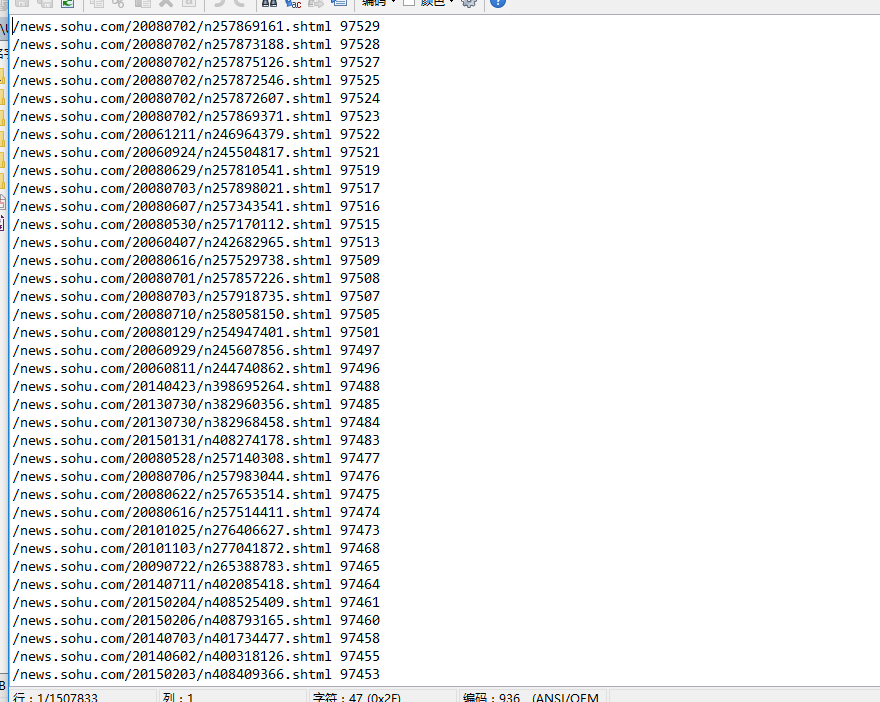
本组程序的编译过程如下图所示：

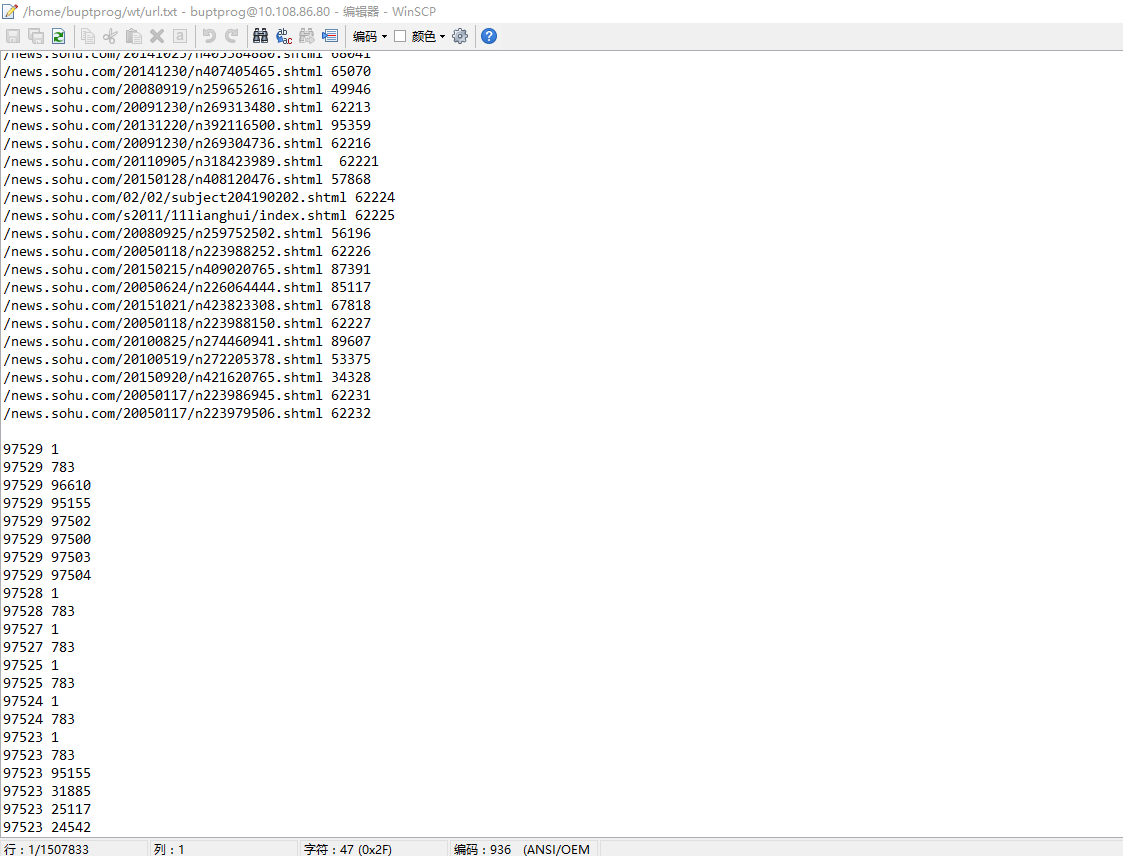
to王丹阳/编译.PNG

本组爬虫部分的运行总时间约为6分钟，下图是爬虫的运行过程：

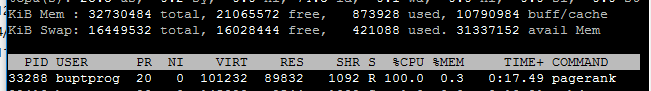
to王丹阳/运行爬虫.PNG

爬虫的结果以及生成的链接关系如下两图所示，爬取到的URL以及编号、对应链接关系编号：





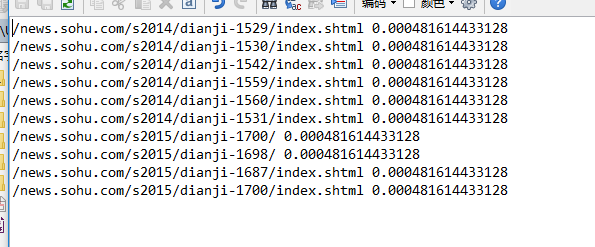
本组程序PageRank部分的运行总时间约为1分钟，运行期间占用的内存约为853.45M，如下图所示：



本组PageRank的运行过程如下图所示：



本组pagerank计算出的TOP10网页结果如下图所示：



* 结论

本组通过此次实验总结了几点注意事项：关于HTTP响应处理，应当只存储200 OK成功响应，其他无视。关于URL去重，只检查某个URL是否已经被抓过了。

本组在生成链接关系的过程中，借助了Bloom过滤器，这点可以作为一个创新点，因为这样大大减少了find的时间，并使得结果更略微准确一些。

* 组内分工

王韬：设计并开发爬虫部分；并负责实验报告中关于系统架构设计部分的书写。

王丹阳：设计并开发PageRank部分；并负责实验报告其他部分的书写。