计算机应用编程实验报告

实验2：网站页面分析

组员：赵孔阳，王浪

1.实验背景及目标

背景：

（1）科研或产品研究都需要从互联网抓取数据

（2）如何设计高性能的网络爬虫，实现高速并发抓取；

（3）垃圾邮件过滤、网页去重等都需要快速去重；

（4）基于图结构分析复杂网络，例如社交网络、科研合作网络、欺诈网络等；

（5）如何利用矩阵来建模图并理解矩阵的作用。

目标：

爬取十几万个页面的网站，构建页面链接网络，计算PageRank最高的前10个页面。

2.系统设计实现

2.1 并行网络爬虫

2.1.1 爬虫整体流程

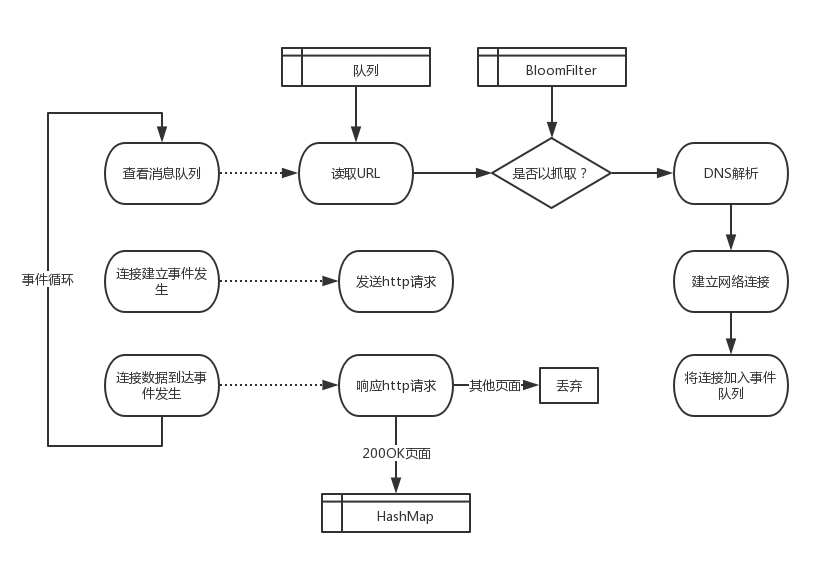
爬虫的整体流程如图2-1所示：

图2-1 爬虫程序整体流程

爬虫程序的流程为：

（1）初始化队列，将第一个页面的URL添加到队列中；

（2）URL出队，建立网络连接；

（3）发送http请求；

（4）响应http请求，无效页面丢弃，有效页面进行编号并解析，URL去重，将未解析过的URL添加到队列中，保存链接关系；解析过的URL直接保存链接关系；

（5）循环（2）-（4）

（6）输出所有有效页面和编号，链接关系转换为编号关系

在实现爬虫的过程中，使用了轻量级的开源高性能事件通知库libevent，通过向libevent框架注册相应的事件和回调函数，当事件发生时，libevent会调用这些回调函数处理相应的事件完成事件处理。

在判断URL是否被爬取过时，使用BloomFilter布隆过滤器过滤掉已经处理过的页面，避免处理过的页面再次添加到队列中，BloomFilter结构如图2-2所示：

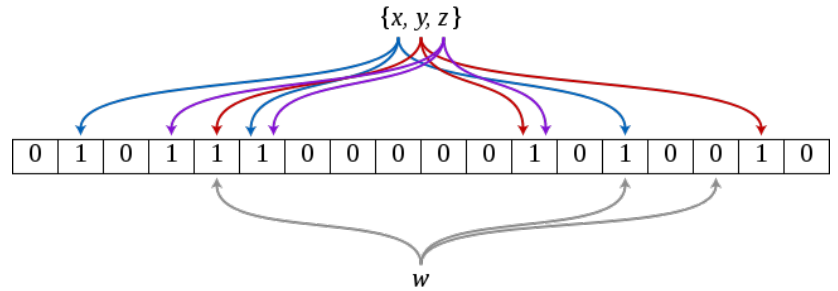


图2-2 BloomFIlter结构

假设有x、y、z三个变量，对于变量x进行不同的hash运算得到不同的值，将下标为该值的数位置为1，对y、z最相同的操作。当有新的变量w时，对变量w进行和x一样的操作，当bit[hash(w)]=0时，说明变量w并未在BloomFilter中出现过，处理变量w，并将其添加到BloomFilter中；若对于每一位bit[hash(w)]=1，说明变量w可能在BloomFilter中出现过，不予处理。

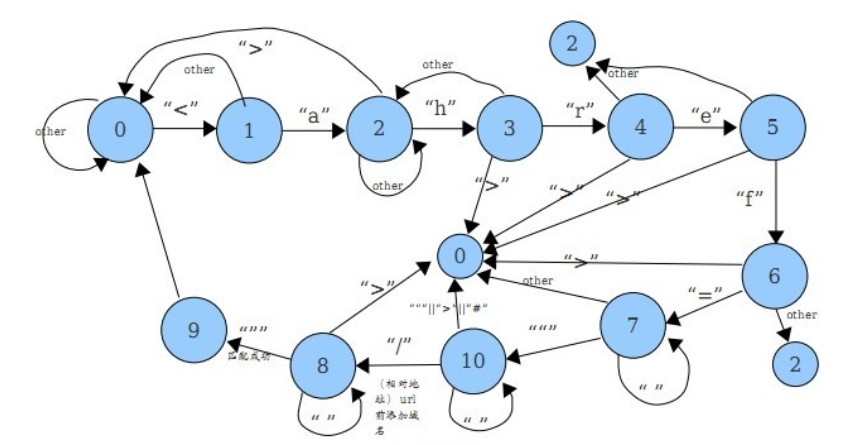
每次接收到从服务器返回的HTML页面后，需要从页面中解析出URL信息，为了实现高效的字符串匹配，我们用到了实验1中使用过的AC自动机，HTML解析使用的AC自动机结构如图2-3所示：

图2-3 用于HTML解析的AC自动机状态转移图

在本次实验中还需要保存URL链接关系，所以需要对URL进行编号，我们通过判断筛选出状态码为200OK的页面，将这样的URL和新的编号添加到HashMap中，并把URL和解析出的URL关系保存到tmp.txt文件中，当队列中所有的URL都处理完后，tmp.txt文本中就保存了所有的URL关系，最后通过HashMap将文本中的URL转成编号，这样得到的结果中就只包含有效页面的关系。

2.1.2 主要数据结构

在事件处理的回调函数中，读事件需要找到关联相同socket套接字的写事件，写事件也需要找到关联相同socket套接字的读事件，所以使用结构体EVENT\_ARG在读写事件之间传递事件指针ev\_w、ev\_r。

其次读写事件回调函数中都会用到相同的URL，所以在EVENT\_ARG中添加char数组，保存当前的URL。

最后，当读事件发生时，服务器发出的响应可能由多个数据包构成，而HTML的状态码只存在第一个数据包中，为了避免重复处理数据包，添加变量firstPack标记第一个数据包，在第一个数据包中查找“200 OK”状态码，找到表明页面有效，继续进行HTML界面解析，若找不到该页面直接丢弃，继续处理下一个URL。EVENT\_ARG结构体如图2-4所示：

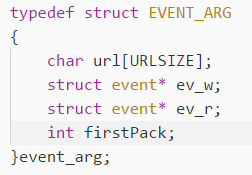


图2-4 EVENT\_ARG结构体

为了保存URL和编号的连接关系，我们使用了开源的HashMap，并设计符合HashMap接口的结构体KEY\_VALUE保存键值对，KEY\_VALUE结构体如图2-5所示：

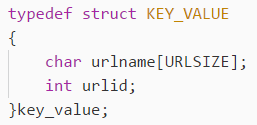


图2-5 KEY\_VALUE结构体

2.2 PageRank算法实现

2.2.1 PageRank算法介绍

Google的创始人之一Larry Page于1998年提出了PageRank，并应用在 Google搜索引擎的检索结果排序上，该技术也是Google早期的核心技术之一。核心思想：（1）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。（2）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。（3） 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。

2.2.2 PageRank算法原理

PageRank算法总的来说就是预先给每个网页一个PR值（下面用PR值指代PageRank值），由于PR值物理意义上为一个网页被访问概率，所以一般是，其中N为网页总数。另外，一般情况下，所有网页的PR值的总和为1。如果不为1的话也不是不行，最后算出来的不同网页之间PR值的大小关系仍然是正确的，只是不能直接地反映概率了，预先给定PR值后，通过下面的算法不断迭代，直至达到平稳分布为止。

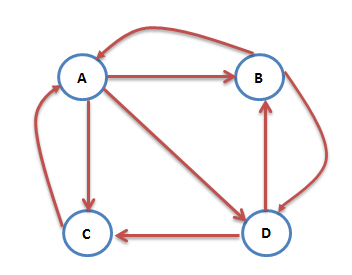


图2-6

图2-6中A的PR值可以表示为：

PR(A) = PR(B) + PR(C)

然而图中除了C之外，B和D都不止有一条出链，所以上面的计算式并不准确。想象一个用户现在在浏览B网页，那么下一步他打开A网页还是D网页在统计上应该是相同概率的。所以A的PR值应该表述为：

PR(A) =

互联网中不乏一些没有出链的网页，如图2-7：

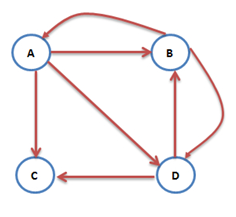


图2-7

图中的C网页没有出链，对其他网页没有PR值的贡献，我们不喜欢这种自私的网页（其实是为了满足 Markov 链的收敛性），于是设定其对所有的网页（包括它自己）都有出链，则此图中A的PR值可表示为：

PR(A) =

然而我们再考虑一种情况：互联网中一个网页只有对自己的出链，或者几个网页的出链形成一个循环圈。那么在不断地迭代过程中，这一个或几个网页的PR值将只增不减，显然不合理。如图2-8中的C网页就是刚刚说的只有对自己的出链的网页：

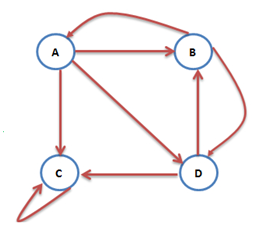
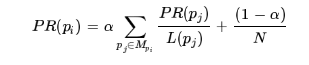


图2-8

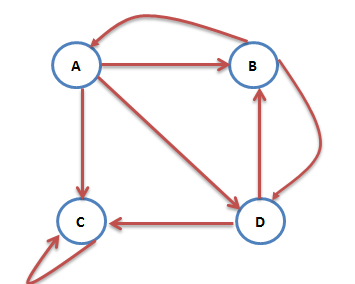
为了解决这个问题。我们想象一个随机浏览网页的人，当他到达C网页后，显然不会傻傻地一直被C网页的小把戏困住。我们假定他有一个确定的概率会输入网址直接跳转到一个随机的网页，并且跳转到每个网页的概率是一样的。于是则此图中A的PR值可表示为：

PR(A) =

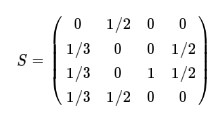
在一般情况下，一个网页的PR值计算如下：



其中Mpi是所有对pi网页有出链的网页集合， L(pj)是网页pj的出链数目，N是网页总数，α一般取0.85。PageRank算法的正确性证明包括上面两点。为了方便证明，我们先将PR值的计算方法转换一下。仍然拿刚刚的例子来说

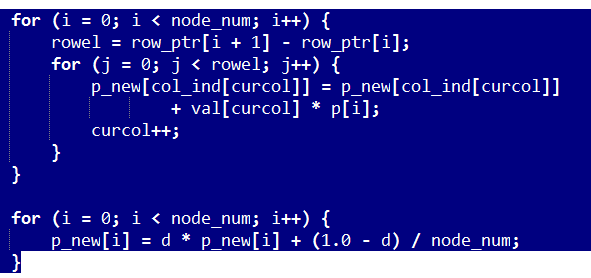


我们可以用一个矩阵来表示这张图的出链入链关系， =0表示j网页没有对i网页的出链：



取e为所有分量都为 1 的列向量,接着定义矩阵：





则PR值的计算如下，其中Pn为第n次迭代时各网页PR值组成的列向量：



于是计算PR值的过程就变成了一个 Markov 过程，那么PageRank算法的证明也就转为证明 Markov 过程的收敛性证明：如果这个 Markov 过程收敛，那么$ \lim\_{n \rightarrow \infty}P\_{n} 存在，且与存在，且与P\_0$的选取无关。若一个 Markov 过程收敛，那么它的状态转移矩阵A需要满足

A为随机矩阵。

A是不可约的。

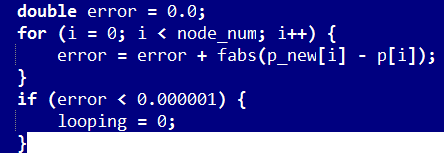
A是非周期的。

先看第一点，随机矩阵又叫概率矩阵或 Markov 矩阵，满足以下条件：显然我们的A矩阵所有元素都大于等于0，并且每一列的元素和都为1。第二点，不可约矩阵：方针A是不可约的当且仅当与A对应的有向图是强联通的。有向图G=(V,E)是强联通的当且仅当对每一对节点对u,v∈V，存在从u到v的路径。因为我们在之前设定用户在浏览页面的时候有确定概率通过输入网址的方式访问一个随机网页，所以A矩阵同样满足不可约的要求。第三点，要求A是非周期的。所谓周期性，体现在Markov链的周期性上。即若A是周期性的，那么这个Markov链的状态就是周期性变化的。因为A是素矩阵（素矩阵指自身的某个次幂为正矩阵的矩阵），所以A是非周期的。

2.2.3 PR值计算方法

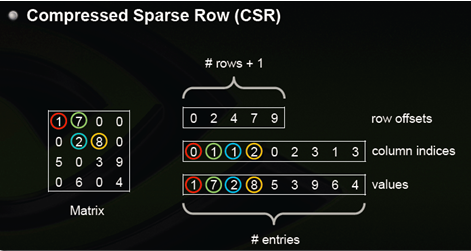
幂迭代法: 首先给每个页面赋予随机的PR值，然后通过=A不断地迭代PR值。当满足下面的不等式后迭代结束，获得所有页面的PR值：

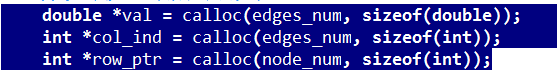


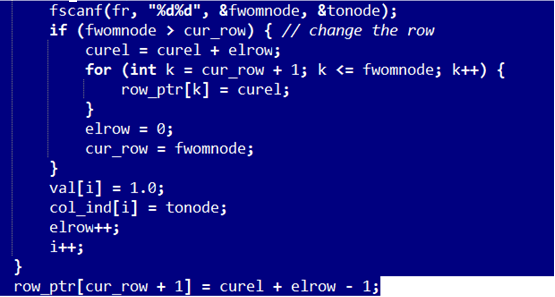


2.2.4 CSR压缩

CSR是比较标准的一种，也需要三类数据来表达：数值，列号，以及行偏移。CSR不是三元组，而是整体的编码方式。数值和列号与COO一致，表示一个元素以及其列号，行偏移表示某一行的第一个元素在values里面的起始偏移位置。如上图中，第一行元素1是0偏移，第二行元素2是2偏移，第三行元素5是4偏移，第4行元素6是7偏移。在行偏移的最后补上矩阵总的元素个数，本例中是9。







3.实验结果

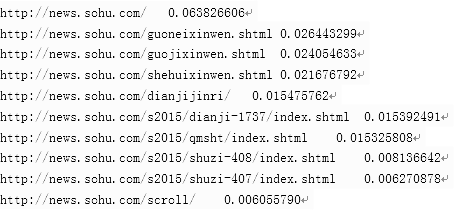
（1）爬虫成功抓取页面108874个；

（2）爬虫获得URL连接关系1550259条；

（3）爬虫程序占用内存最大时约为650MB；

（4）爬虫程序运行时间为812s；

（5）PageRank算法求得Top10的网站及rank值为：



4.优化、不足及遇到的问题

4.1 优化

在实现爬虫程序的过程中，起初我们发现程序内存占用高达1.2G，检查代码发现在很多地方没有及时释放内存，修改后内存峰值降至650MB；在第一版程序编写完时发现耦合度太高，导致代码冗余，修改难度大，于是我们重新编写，尽可能保证功能分离，每个文件实现单一功能或解决同一类问题，这样在爬虫主程序中代码清晰，也有利于后续修改。

在实现PageRank算法过程中，计算PageRank值不能用传统线性代数方法，由于PageRank值是特征值为一的特征向量，迭代计算可以很方便得到。连接关系的存储结构设计是快速计算的前提，若用普通矩阵存储数据，不仅占用内存空间大，计算速度也很慢（计算了矩阵中的每一个元素，包括0），链接关系图的边数较少，用稀疏矩阵存储可以接受极大空间，计算是只需要计算非零元素，2秒就可以搞定整个PageRank值的计算。

4.2 不足

整体代码完成后仍存在很多不足：

（1）程序整体结构、内存申请释放仍有可调整的地方；

（2）DNS解析过程使用系统函数完成，可尝试异步解析

（3）传输时加入zlib压缩

（4）由于libevent是异步机制，爬虫程序是单线程完成，很容易在外部造成阻塞

4.3 遇到的问题

（1）对于每个HTML页面而言，解析得到的URL包括解析过的URL也包括未解析的URL，前者不需要加入队列，但需要保存链接关系，后者需要加入队列但不能在此时编号，因为未解析的URL可能无法正常访问，所以我们先把访问成功的URL及编号放到HashMap中，把链接关系写入tmp.txt文件中，等待所有URL处理完后，再将URL替换为编号，这样就避免了失效页面也被编号的问题。

（2）在使用libevent的过程中，发现注册事件、添加事件、事件回调时需要用到一些数据，但是libevent是异步处理的，所以通过传递单一变量是不够的，于是我们设计了EVENT\_ARG结构体用于参数传递。

（3）在HTML解析过程中相同的连接关系可能会出现多次，为了避免这种情况，我们在每次解析HTML时创建新的HashMap过滤掉重复出现的URL，在解析结束后释放掉该HashMap，这样获取到的链接关系就不会重复了，也有利于后面PageRank的计算。

5.收获

实验二的难度明显要大于实验一，在实现爬虫程序的过程中，从简单的socket程序开始，实现URL去重、完成HTML解析、使用libevent完成事件处理，解决URL编号问题、实现PageRank算法等让我们感受到了这次实验的难点所在，也从学习知识点和遇到的问题中发现了自己的不足，特别是解决问题的能力仍需提高，在以后的工作和学习中，这种能力的培养也是我们需要重视的。