**作业报告**

**一、输入&输出**

原始数据：维基百科的xml文件以及index文件：

enwiki-20181001-pages-articles-multistream.xml

enwiki-20181001-pages-articles-multistream-index.txt

输出数据：排序后的pageRank值文件（每一行表示一个页面标题（概念）以及pagerank值，格式为“概念 \t PageRank值”）：

result.txt

**二、处理流程：**

1.根据index文件建立从页面标题到id值以及从id值到页面标题的字典

2.解析XML文件，取170万个页面，解析页面的链接

3.剔除不指向该170万样本页面的链接，分析页面间的链接关系，生成有向图

4.根据有向图，迭代计算每个页面的pageRank值

5.对计算结果进行排序，输出排名前100万页面的相关数据

关于字典的用法：

有两对字典（即一共四“本”字典），第一对字典是从页面标题和页面id之间的字典,第二对字典是从页面id和计算pageRank值时数组下标的映射，引入两对字典都是为了在计算时提高运行速度。第一对字典用于解析XML文件时，用id直接表示页面间的链接关系。第二对字典是在计算pageRank时，程序用数组来存储每个页面的被连接的情况和页面的pageRank值，为了减少反复索引起的资源浪费，这里将页面id映射到数组相应的下标，然后再计算PageRank值的时候直接用每个页面对应的数组下标来代替之前的页面id。

**三、处理流程简要介绍**

1.根据index文件建立从页面标题到id值，以及从id值到页面标题的字典

[源代码中的parseIndex.py实现]

Index文件中的每个条目都是以:

Num->[0-9]+

PageName->.+

Item->Num:Num:PageName\n

这样的正则定义呈现的，因此这里直接按照’\n’对index文件进行分割，再剔除前面的两个数字和冒号，然后对每个页面从0开始标Id（int类型），生成页面标题到id和id到页面标题的两“本”字典，存入本地磁盘。

2.解析XML文件，取170万个页面，解析页面的链接

[源代码中的parseXml.py实现]

观察XML文件的内容可以知道，有三类数据是我们需要的：

1. 是”<page>”和”</page>”，这两个字符串，它们标示着一个页面的开始和结束。
2. 是”<title>”和”</title>”之间的字符串，这就是该页面的标题。
3. 就是”[[“和“]]”之间的字符串以及”redirect title=”之后跟随的字符串，这些字符串就是被解析页面所链接到的页面名

由于XML文件较大，这里采用按行读取文件，读到”<page>”进入一个循环，直到读取到”</page>”，将循环中读取的字符串存入一个string类中，直接用正则表达式匹配找出：页面标题、页面链接到的其他页面的标题。特别的是，第三类数据可能会有”PageName\_1|PageName\_2”，即用’|’分割开的页面名，这里直接用’|’分割该字符串，然后提取出PageName\_1和PageName\_2。

最后，用第一步得到的从页面标题到页面id的字典对以上得到的所有页面标题进行映射并将结果存入磁盘。

3.剔除不指向该170万样本页面的链接，分析页面间的链接关系，生成有向图

[源代码中的transform.py实现]

提取出的170万样本页面的id，写入一个数组中。排序。然后对每个页面中链接到其他页面的id们（即页面的“出边”），逐一在这个数组中进行二分查找，如果找不到就剔除该“出边”。

统计每个页面的“出边”数目，并将结果计入一个数组当中，存入本地磁盘。

之后就是“将出边转换为入边”，首先为每个页面建立各建立一个新数组。然后解析之前每个页面的“出边”情况，遍历每个页面（这里假定为first页面）的出边序列，根据序列中的id找到相应新数组，将first页面的id添加到相应新数组中，重复直至处理完所有数据。

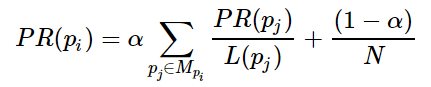
最后得到一个有向图，图的顶点集是该170万样本页面id的集合，每个id都对应着一个“入边”数组，数组中的元素就是链接到该页面的页面们的id。将170个样本页面id和相应的“入边”数组们存入本地磁盘。

4.根据有向图，迭代计算每个页面的PageRank值

[源代码中的pageRank.cpp实现]

为了减少计算PageRank值时反复索引导致的资源浪费，这里再次使用字典将170万样本页面的id映射为从0到170万的连续数字。

关于一个页面的迭代运算的公式如下所示：



其中Mpi是所有链接到pi的网页集合，L(pj)是网页pj的出边数目，N是网页总数，α在这里取0.85。

在迭代计算之前，检查每一个页面节点的出边数（这个在第三步已经存储在一个数组当中），针对没有出边的节点，让其变成对所有的节点都有出边。之后，对每个节点赋予初始的PageRank值为1/N。

然后开始迭代。在每次迭代过程中，遍历每一个节点，对每一个节点都计算出新的PageRank值，并将新值存储在一个临时数组当中。迭代直至收敛，即所有节点PageRank值更改量的绝对值之和小于某个非常小的常数。

最后将计算出的PageRank数组存入本地磁盘。

5.对计算结果进行排序，输出数据

[源代码中的last.py实现]

从磁盘中读入PageRank数组，并使用 数组下标->页面id 和 页面id->页面标题 两“本”字典得出每个页面标题与PageRank值的对应，对最终结果进行排序，将排序结果写入本地磁盘。

**四、关于PageRank算法的思考**

PageRank算法的关键问题是是否存在，其中表示第n次迭代后所有页面PageRank值组成的列向量。用方形矩阵S表示页面间的关系，其中：若j网页对i网页没有出链，则；若j网页对i网页有出链，则, 是j网页的出边数目。取e为所有分量都为 1 的列向量,接着定义一个矩阵,则。因此计算PageRank值的过程就变成了一个Markov过程。根据Markov过程的性质，该算法要收敛，必须满足三个条件：U的每一列元素和要为1，U是不可约的，U是非周期的。

第一个条件，必须特殊处理那些没有出边的网页，否则某个没有出边的网页在A中对应的列向量元素和会变成1-α，不等于1。因此在程序中的算法就默认让没有出边的页面变成对所有页面都有出边（包括它自己），这样也满足了PageRank算法的初衷（当用户浏览到一个没有外部链接的网页，他会倾向于输入网址直接跳转到一个随机的网页）。

第二个条件，由于之前设定了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，因此与U对应的有向图是强联通的，从而U是不可约的。

第三个条件，因为U是素矩阵，其某个次幂为正矩阵，所以U是非周期的。