PageRank 用户手册

PageRank User’s Manual

目录

1 算法简介[1]

PageRank 算法以谷歌创始人之一 Larry Page 命名，在谷歌搜索中被用来对搜索结果进行排名调整，是一种度量网页重要性的方法。以下引自谷歌：

> PageRank 根据一个网页入链的数量和质量估算它的重要性。假定一个网页越是重要，它的入链数量就会越多。

PageRank 不是谷歌用来调整结果排名的唯一一个算法，但它最先被使用而且最为有名。

PageRank 算法输出一个概率分布，用来表示一个人在一个网页上随机点击一系列链接，最终到达某个页面的可能性。它可以被用来计算任意大小的文件集。假定在计算开始时概率分布均匀，数次迭代计算后 PageRank 值将接近真实情况。

简化算法

假设集合里包含 A, B, C, D 四个网页，没有指向自身的链接，而且从一个网页指向另一个的链接最多有一条。所有页面的 PageRank 值都初始化为相同值（早期版本的 PageRank 中初始值为 1，后来为 0 和 1 之间的一个值，下面的例子中为 0.25）[2]。

一个网页的 PageRank 值平均分配给它的每一条出链，用来更新该出链所指向页面的 PageRank 值。

假如系统中只存在分别从 B, C, D 指向 A 的链接，在下一轮迭代中，每条链接将传递给 A 0.25 的 PageRank 值，总共 0.75：

$$PR(A) = PR(B) + PR(C) + PR(D).$$

假如 B 指向 C 和 A，C 指向 A，D 指向 A，B 和 C。第一轮迭代中，B 将传递 0.125 给 A，另外 0.125 给 C，C 传递 0.25 给 A，D 分别传递大约 0.083 给其它 3 个页面。A 的 PageRank 值变为约 0.458：

$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{1} + \frac{PR(D)}{3}.$$

即出链传递的 PageRank 值等于页面的 PageRank 值除以出链数量 L( )：

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}.$$

一般地，网页 u 的 PageRank 值的表达式为：

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)},$$

这里 B_u 是有链接指向页面 u 的所有页面的集合。

[1] 译自 PageRank – Wikipedia: https://en.wikipedia.org/wiki/PageRank

[2] 本算法中该值为 1

阻尼系数

PageRank 理论中假想的在网上随机点击链接的人，最终会停止点击。在每一步中，阻尼系数 d 就是他继续点击的概率。这个概率值一般假定是 0.85 左右。

1 减去阻尼系数（在一些版本的算法中[3]还要除以页面总数 N）后，加上入链传递的 PageRank 值与阻尼系数的乘积，得到新的 PageRank 值：

$$PR(A) = \frac{1-d}{N} + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right).$$

最终表达式如下：

$$\mathbf{R} = \begin{bmatrix} (1-d)/N \\ (1-d)/N \\ \vdots \\ (1-d)/N \end{bmatrix} + d \begin{bmatrix} \ell(p_1,p_1) & \ell(p_1,p_2) & \cdots & \ell(p_1,p_N) \\ \ell(p_2,p_1) & \ddots & & \vdots \\ \vdots & & \ell(p_i,p_j) & \\ \ell(p_N,p_1) & \cdots & & \ell(p_N,p_N) \end{bmatrix} \mathbf{R}$$

其中，如果没有 p_j 到 p_i 的链接，$\ell(p_i,p_j)$ 为 0。对每个 *j* 有

$$\sum_{i=1}^{N} \ell(p_i,p_j) = 1$$

关于算法更多的细节，参见 https://en.wikipedia.org/wiki/PageRank

[3] 例如本算法

1 An intro of the algorithm[4]

PageRank is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google:

> PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

It is not the only algorithm used by Google to order search engine results, but it is the first algorithm that was used by the company, and it is the best-known.

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called "iterations", through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

Simplified algorithm

Assume a small universe of four web pages: A, B, C and D. Links from a page to itself, or multiple outbound links from one single page to another single page, are ignored. PageRank is initialized to the same value for all pages. In the original form of PageRank, the sum of PageRank over all pages was the total number of pages on the web at that time, so each page in this example would have an initial value of 1. However, later versions of PageRank, and the remainder of this section, assume a probability distribution between 0 and 1. Hence the initial value for each page in this example is 0.25. [5]

The PageRank transferred from a given page to the targets of its outbound links upon the next iteration is divided equally among all outbound links.

If the only links in the system were from pages B, C, and D to A, each link would transfer 0.25 PageRank to A upon the next iteration, for a total of 0.75.

$$PR(A) = PR(B) + PR(C) + PR(D).$$

Suppose instead that page B had a link to pages C and A, page C had a link to page A, and page D had links to all three pages. Thus, upon the first iteration, page B would transfer half of its

[4] Select from PageRank – Wikipedia: https://en.wikipedia.org/wiki/PageRank
[5] In this algorithm, this value is 1

existing value, or 0.125, to page A and the other half, or 0.125, to page C. Page C would transfer all of its existing value, 0.25, to the only page it links to, A. Since D had three outbound links, it would transfer one third of its existing value, or approximately 0.083, to A. At the completion of this iteration, page A will have a PageRank of approximately 0.458.

$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{1} + \frac{PR(D)}{3}.$$

In other words, the PageRank conferred by an outbound link is equal to the document's own PageRank score divided by the number of outbound links L( ).

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}.$$

In the general case, the PageRank value for any page u can be expressed as:

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)},$$

i.e. the PageRank value for a page u is dependent on the PageRank values for each page v contained in the set Bu (the set containing all pages linking to page u), divided by the number L(v) of links from page v.

Damping factor

The PageRank theory holds that an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue is a damping factor d. Various studies have tested different damping factors, but it is generally assumed that the damping factor will be set around 0.85.

The damping factor is subtracted from 1 (and in some variations of the algorithm, the result is divided by the number of documents (N) in the collection[6]) and this term is then added to the product of the damping factor and the sum of the incoming PageRank scores. That is,

$$PR(A) = \frac{1-d}{N} + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right).$$

The final equation is as follow,

[6] This algorithm is a case

$$\mathbf{R} = \begin{bmatrix} (1-d)/N \\ (1-d)/N \\ \vdots \\ (1-d)/N \end{bmatrix} + d \begin{bmatrix} \ell(p_1,p_1) & \ell(p_1,p_2) & \cdots & \ell(p_1,p_N) \\ \ell(p_2,p_1) & \ddots & & \vdots \\ \vdots & & \ell(p_i,p_j) & \\ \ell(p_N,p_1) & \cdots & & \ell(p_N,p_N) \end{bmatrix} \mathbf{R}$$

where the adjacency function $\ell(p_i,p_j)$ is 0 if page p_j does not link to p_i, and normalized such that, for each *j*

$$\sum_{i=1}^{N} \ell(p_i,p_j) = 1$$

More details of the algorithm can be found at https://en.wikipedia.org/wiki/PageRank

2 C 语言实现

PageRank.c 里是 C 语言实现的源代码。

2.1 函数库

stdio.h

stdlib.h

unistd.h

chdir(const char * path)函数，切换工作目录至 path，用于输出。[7]

2.2 宏

CEILING

乘法计算的最多迭代次数，默认为 80[8]。

P

PageRank 阻尼系数，默认为 0.85。

OK

函数调用成功的返回值，默认为 1。

ERROR

函数调用失败的返回值，默认为 0。

2.3 数据结构

status

函数调用状态。

page

网页信息，包含 name 域（网页名称）和 value 域（PageRank 值）。

Edge

矩阵基本元素，存储 web graph 中的一条边，包含数据域 x（终点），y（起点），value（权重）和指针域 Edge *right（同一终点的下一条边），Edge *down（同一起点的下一条边）。

List

一个 Edge 的指针。

---

[7] 见 http://c.biancheng.net/cpp/html/305.html，注：windows 里没有 unistd.h 这个库

[8] 见第 4 部分注 2

CrossList

十字链表存储概率转移矩阵，包含指针域 List *RHead（矩阵行的头结点），List *CHead（矩阵列的头结点）。

2.4 函数[9]

status Pagerank(char *filename, int num_pages, char *dir, int threads);

由 main 函数调用，负责数组初始化，调用其他函数，打印状态，输出结果。

status CreateCL(CrossList *M, char *filename, int num_pages);

由 Pagerank 函数调用，创建矩阵，读取文件进行初始化。

status RInsert(CrossList *M, Edge *p, int i);

由 CreateCL 函数调用，行插入。

status CInsert(CrossList *M, Edge *p, int j);

由 CreateCL 函数调用，列插入。

status DestroyCL(CrossList *M);

由 CreateCL 函数调用，销毁创建前已存在的矩阵。

status Cal_pt(CrossList *pt, int num_pages, double random);

由 Pagerank 函数调用，计算概率转移矩阵，暂时忽略边不存在时随机跳转的情况。

status Calculation(CrossList *pt, page *pr1, page *pr2, int threads, int num_pages, double threshold, double random);

由 Pagerank 函数调用和自身递归调用，调用 Mul 函数计算矩阵-向量乘法，判断计算结果是否达到阈值或迭代次数是否达到最大值。

status Mul(CrossList *pt, page *pr1, page *pr2, int threads, int num_pages, double random);

由 Calculation 函数调用，计算矩阵-向量乘法。

status MinMax(page *pr1, int *min_name, double *min_value, int *max_name, double *max_value, int num_pages);

由 Pagerank 函数调用，得到 PageRank 值最大和最小的网页。

2.5 OpenMP 并行计算

算法中 99%以上的时间花费在矩阵-向量乘法上，因此只有这个部分有必要进行并行计算。Mul 函数中加入一条 pragma 语句（第 298 行），线程数由变量 threads 指定（命令行参数之一[10]）。在不支持 OpenMP 的编译器中，这条语句会被自动忽略，程序串行执行，不影响其正确性。

[9] 注：文件操作函数 fopen, fscanf 在 vs 2015 中会报错，要求使用 fopen_s, fscanf_s

[10] 见 3.2

2.6 输入文件格式

输入文件开头部分可以有若干以#开头的注释。[11]

Web graph 中存在一条由 a 指向 b 的边，在输入文件中记为

b a

输入文件可按 BigDataBench 用户手册中 PageRank 部分的步骤由 Hadoop 生成。

[11] 若数据中间部分或结尾出现注释，需要对 CreateCL 函数作简单修改

2 Algorithm in C language

Source code is in PageRank.c.

2.1 Function libraries

stdio.h
stdlib.h
unistd.h
function chdir(const char * path) changes the work directory to path to output.[12]

2.2 Macros

CEILING
The maximum of iterations, default 80[13]
P
PageRank dumping factor, default 0.85。
OK
The return value of a successful call, default 1
ERROR
The return value of a unsuccessful call, default 0

2.3 Data structure

status
The status of a function call
page
Information of a page, consisting of a name field (the name of the page) and a value field (the PageRank value of the page)
Edge
Basic element of the matrix, storing a edge of the web graph, consisting of data fields x (ending point), y (starting point) and pointer fields right (the next edge with same ending point), down (the next edge with same starting point)
List
A pointer to an Edge
CrossList

[12] More details: http://c.biancheng.net/cpp/html/305.html, remark: there is no unistd.h in Windows
[13] Pay attention to the remark 2, Part 4

A cross list used to store the probabilistic transfer matrix, consisting of pointer fields RHead (head nodes of rows in a matrix), CHead (head nodes of columns in a column)

2.4 Functions[14]

status Pagerank(char *filename, int num_pages, char *dir, int threads);

Called by function main, to initialize vectors, call other functions, print statuses and output the results

status CreateCL(CrossList *M, char *filename, int num_pages);

Called by function Pagerank, to create a matrix and initialize it

status RInsert(CrossList *M, Edge *p, int i);

Called by function CreateCL, to insert in a row

status CInsert(CrossList *M, Edge *p, int j);

Called by function CreateCL, to insert in a column

status DestroyCL(CrossList *M);

Called by function CreateCL, to destroy a matrix that already exists

status Cal_pt(CrossList *pt, int num_pages, double random);

Called by function Pagerank, to calculate probabilistic transfer matrix, random jump without an edge is ignored for now

status Calculation(CrossList *pt, page *pr1, page *pr2, int threads, int num_pages, double threshold, double random);

Called by function Pagerank and Calculation itself, to call function Mul, and judge whether the result has reached the threshold or the number of iteration has reached the maximum

status Mul(CrossList *pt, page *pr1, page *pr2, int threads, int num_pages, double random);

Called by function Calculation, to calculate multiplication between a matrix and a vector

status MinMax(page *pr1, int *min_name, double *min_value, int *max_name, double *max_value, int num_pages);

Called by function Pagerank, to get maximum and minimum of PageRank values

2.5 OpenMP parallel calculation

The matrix-vector multiplication takes over 99% of the time so it's the only part that needs to be parallelized. A pragma sentence is added in function Mul (Line 298). The number of threads is appointed by variable *threads*, which is one of the command line parameters[15]. When the source code is compiled in a compiler that doesn't support OpenMP, this sentence is ignored

[14] Remark: functions fopen and fscanf cause error in vs 2015, and fopen_s and fscanf_s are required to use

[15] More details is provided at 3.2

automatically. The program will execute serially and correctly.

2.6 The format of input file

Annotations start with a '#' is allowed at the beginning of the input file.[16]
If there is an edge from *a* to *b*, denoted as

b a

in input file.
The input file can be generated in Hadoop by following PageRank part in BigDataBench user's manual

[16] A simple modification of function CreatCL is needed if annotations appear at the middle or ending part of input file.

3 使用方法

3.1 编译

gcc PageRank.c -o pagerank -fopenmp
参数-fopenmp 允许在程序中加入 OpenMP 特性。

3.2 命令行参数

除去程序名本身(argv[0])外，共有 4 个命令行参数。
argv[1]
　　输入文件，包括完整的目录和文件名，缺省目录(./)为工作目录。
argv[2]
　　网页数量。
argv[3]
　　输出目录，缺省目录(./)为工作目录。
argv[4]
　　并行线程数。

3.3 运行

./pagerank filename num_pages dir threads

3 Usage

3.1 Compilation

gcc PageRank.c -o pagerank -fopenmp
Parameter –fopenmp allows OpenMP in program.

3.2 Command line parameters

There are 4 parameters except the name of this program:
argv[1]
Input file, include full directory and file name, default directory (./): work directory
argv[2]
The number of pages
argv[3]
Output directory, default (./): work directory
argv[4]
The number of parallel threads

3.3 Execution

./pagerank filename num_pages dir threads

4 测试

服务器：hw112，2*6 核 CPU
测试文件：Google_genGraph_16.txt
线程数：12
输出结果：output
性能监测：a.xls
执行时间：245s，CPU 利用率：97%

注：

1.程序接口与 Hadoop 上的 Java 版本不完全相同，Java 版本中部分参数在 shell 脚本中指定，输出的 PageRank 值分散在不同的文件中。

2.计算结果与 Java 版本有出入，原因未知。宏 CEILING 的默认值是在本测试中最接近 Java 结果的值。

4 Test

Server: hw112, 2 * 6-core CPU
Test file: Google_genGraph_16.txt
Number of parallel threads: 12
Output directory: output
Performance monitoring: a.xls
Execution time: 245s, CPU rate: 0.97

Remarks:

1. The interfaces are not exactly the same as Java-version algorithm in Hadoop. In Java-version algorithm, several parameters are appointed in the shell script, and the output of PageRank values are placed in several files instead of one.

2. The calculation result is different from Java-version algorithm for an unknown reason. The default of macro CEILING contributes to a most-approximate result to Java-version algorithm.