基于MapReduce的搜索引擎算法

图

- 搜索引擎
- 社交网络
- 路径规划
- ……

图问题与MapReduce

- 一些图问题：
  - 最短路径
  - 最小生成树
  - 广度优先搜索
  - PageRank
- 关键问题：
  - 怎么在MapReduce中表示图数据
  - 怎么用MapReduce遍历图

图表示方法

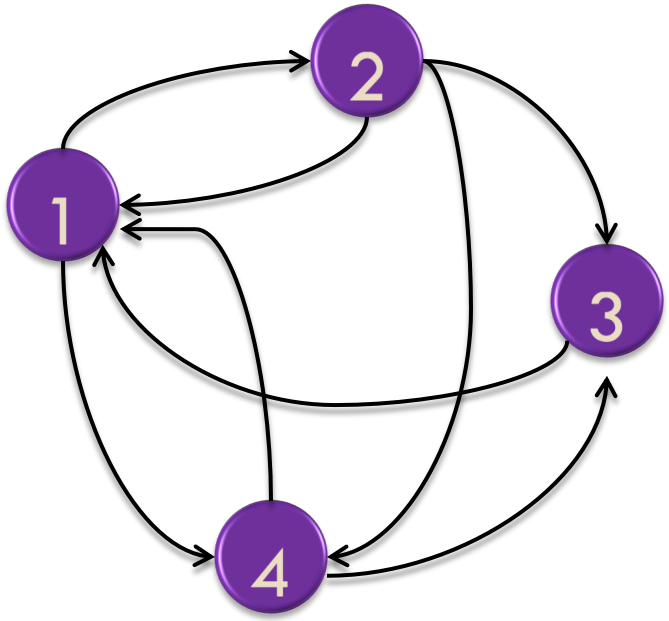
- G = (V, E)
- 两种常见的表示方法
  - 邻接矩阵
  - 邻接表

邻接矩阵

用一个 $n \times n$ 的矩阵 M 来表示图：

- $n = |V|$
- $M_{ij} = 1$ 表示一条i到j的边

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0



邻接矩阵

- 优点：
  - 便于数学操作
  - 遍历一行可得到出度信息，遍历一列可以得到入度信息
- 缺点：
  - 对于稀疏矩阵浪费内存

邻接表

- 只保存邻接矩阵中不为零的节点

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

邻接表

- 优点：
  - 表示更加紧凑
  - 容易得到出度信息
- 缺点：
  - 不方便得到入度信息

PageRank内容概述

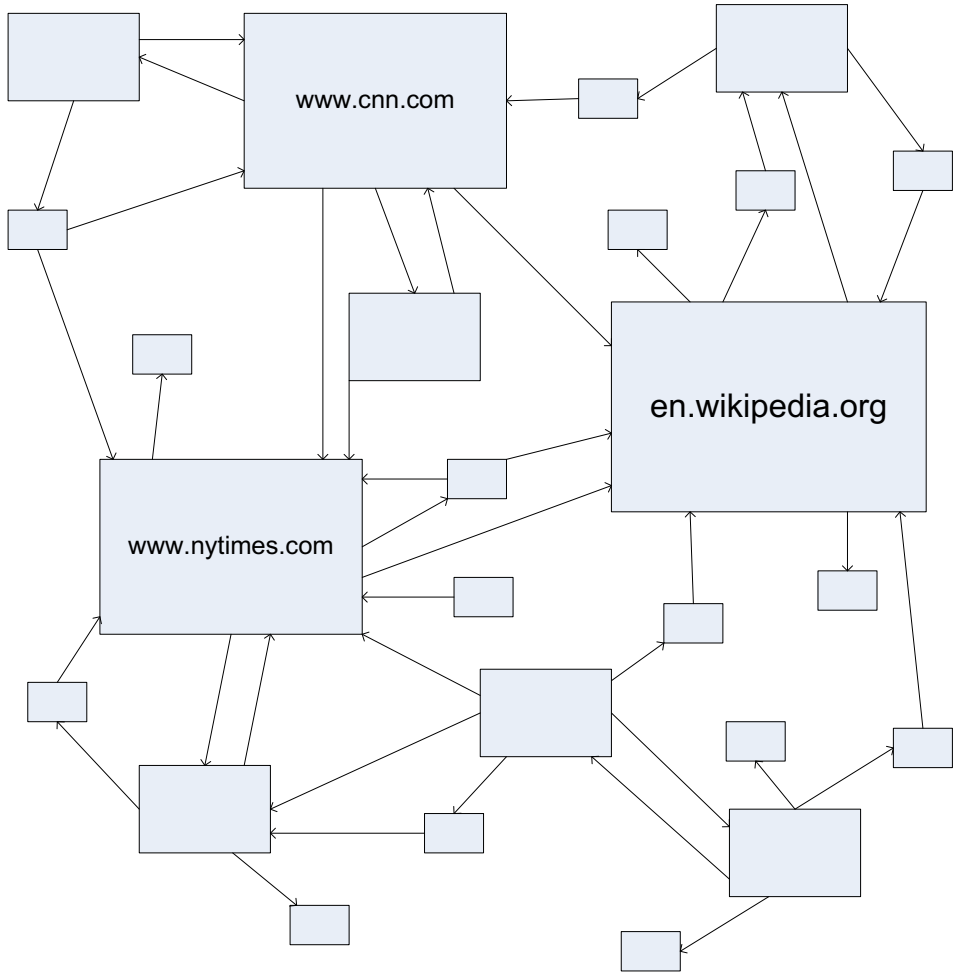
- 什么是PageRank
- PageRank的简化模型
- PageRank的随机浏览模型
- PageRank的MapReduce实现

什么是PageRank

- **PageRank**
  - PageRank是一种由搜索引擎根据网页之间相互的超链接计算的网页排名技术。
  - Larry Page and Sergey Brin, Google, 1997
  - PageRank是Google用于用来标识网页的等级或重要性的一种方法。其级别从1到10级，PR值越高说明该网页越受欢迎（越重要）。

PageRank的基本设计思想和设计原则

- 从许多优质的网页链接过来的网页，必定还是优质网页。一个网页要想拥有较高的PR值的条件：
  - 有很多网页链接到它；（数量假设）
  - 有高质量的网页链接到它。（质量假设）



PageRank的基本设计思想和设计原则

- 如果网页T存在一个指向网页A的连接，则表明T的所有者认为A比较重要，从而把T的一部分重要性得分赋予A。这个重要性得分值为：PR(T)/L(T)
  - 其中PR(T)为T的PageRank值，L(T)为T的出链数
  - 则A的PageRank值为一系列类似于T的页面重要性得分值的累加。

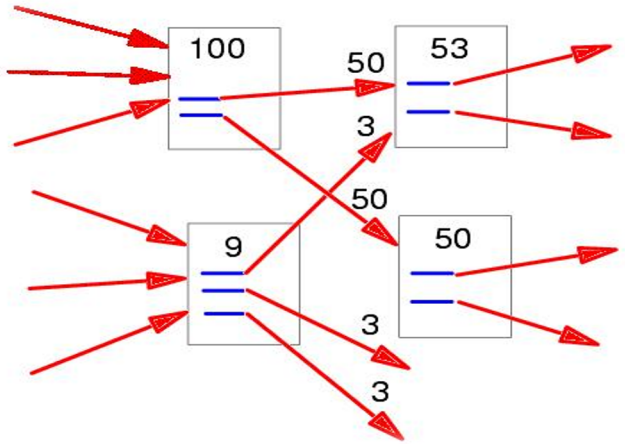
- 即一个页面的得票数由所有链向它的页面的重要性来决定，到一个页面的超链接相当于对该页投一票。一个页面的PageRank是由所有链向它的页面（链入页面）的重要性经过递归算法得到的。一个有较多链入的页面会有较高的等级，相反如果一个页面没有任何链入页面，那么它没有等级。

PageRank的简化模型

- 可以把互联网上的各个网页之间的链接关系看成一个有向图。
- 对于任意网页Pi，它的PageRank值可表示为：

$$R(P_i) = \sum_{P_j \in B_i} \frac{R(P_j)}{L_j}$$

其中Bi为所有链接到网页i的网页集合，
Lj为网页j的对外链接数（出度）。



简化模型的矩阵表示

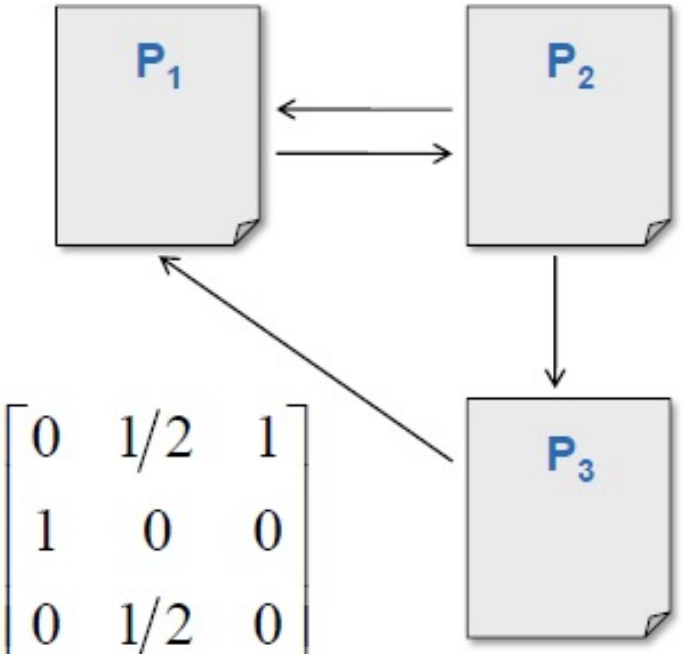
- Let us define a hyperlink matrix **H**

$$\mathbf{H}_{ij} = \begin{cases} 1/L_j & \text{if } P_j \in B_i \\ 0 & \text{otherwise} \end{cases}$$

and $\mathbf{R} = [R(P_i)]$

$\rightarrow \mathbf{R} = \mathbf{HR}$

R is an eigenvector of **H** with eigenvalue 1



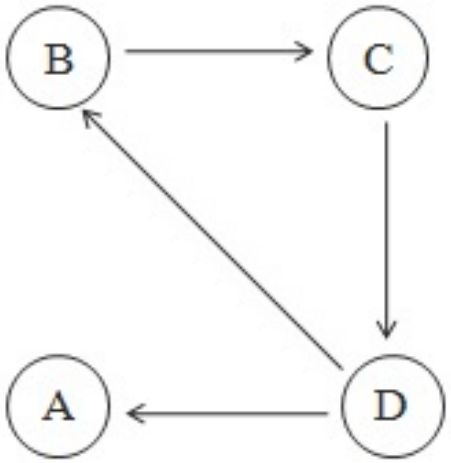
$$\mathbf{H} = \begin{bmatrix} 0 & 1/2 & 1 \\ 1 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix}$$

R为特征向量，H为初始转移矩阵，比如第一列表示从网页P1跳转到其他页面的概率

简化模型面临的问题

- 实际的网络超链接环境没有这么理想化，PageRank会面临两个问题：
  - 排名泄露（Rank leak）
  - 排名下沉（Rank sink）

Rank Leak



	PR(A)	PR(B)	PR(C)	PR(D)
初始	0.25	0.25	0.25	0.25
一次迭代	0.125	0.125	0.25	0.25
二次迭代	0.125	0.125	0.125	0.25
三次迭代	0.125	0.125	0.125	0.125
…	…	…	…	…
n次迭代	0	0	0	0

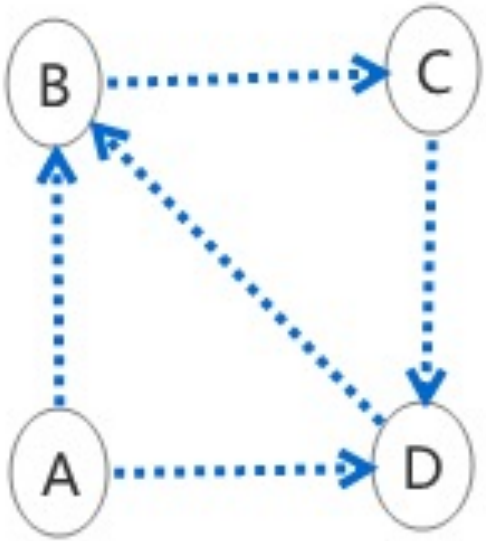
Rank leak：一个独立的网页如果没有外出的链接就产生排名泄漏（多次迭代后，所有网页的PR值都趋向于0）。

解决办法：

1.将无出度的节点递归地从图中去掉，待其他节点计算完毕后再添加。

2.对无出度的节点添加一条边，指向那些指向它的顶点。

Rank Sink



	PR(A)	PR(B)	PR(C)	PR(D)
初始	0.25	0.25	0.25	0.25
一次迭代	0	0.375	0.25	0.375
二次迭代	0	0.375	0.375	0.25
三次迭代	0	0.25	0.375	0.375
四次迭代	0	0.375	0.25	0.375
五次迭代	0	…	…	…

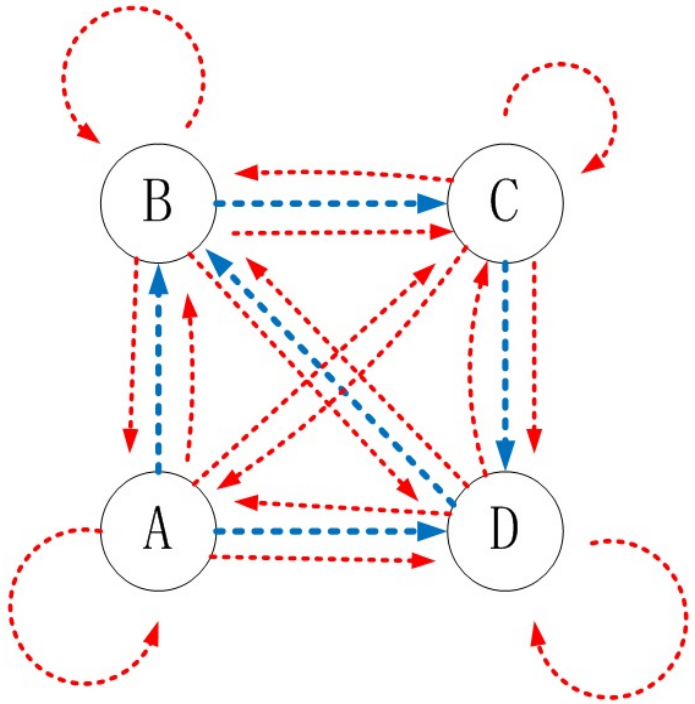
Rank sink：若网页没有入度链接（如节点A），其所产生的贡献会被整个网页图中的一组紧密链接成环的网页（如B、C、D）“吞噬”掉，节点A的PR值在迭代后会趋于0。

解决方法：引入随机浏览模型。

PageRank的随机浏览模型

- 假定一个上网者从一个随机的网页开始浏览
- 上网者不断点击当前网页的链接开始下一次浏览。
- 但是，上网者最终厌倦了，开始了一个随机的网页。
- 随机上网者访问一个新网页的概率就等于这个网页的PageRank值。因此这个模型更加接近于用户的行为。

随机浏览模型的图表示



设定任意两个顶点之间都有直接通路，在每个顶点处以概率d按原来蓝色方向转移，以概率1-d按红色方向转移。

随机浏览模型的矩阵表示

- 回顾简单模型的矩阵表示：
  - R = HR
- 随机浏览模型的矩阵表示：
  - 令： H' = d*H + (1-d)*$[1/N]_{N\times N}$
  - 则： R= H'R
  - 其中R为列向量，代表PageRank值；H'代表转移矩阵；d代表阻尼因子，通常设为0.85。
- 由于等式R=H'R满足马尔可夫链的性质，如果马尔可夫链收敛，则R存在唯一解

随机浏览模型的邻接表表示

- 由于网页数目巨大，网页之间的连接关系的邻接矩阵是一个很大的稀疏矩阵。
- 采用邻接表来表示网页之间的连接关系。
- 随机浏览模型的PageRank公式：

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

  - 通过迭代计算得到所有节点的PageRank值。

随机浏览模型

- 随机浏览模型的优点：
  - 更加符合用户的行为
  - 一定程度上解决了rank sink问题
  - 保证PageRank存在唯一值。

用MapReduce实现PageRank

- Phase1: GraphBuilder
  - 建立网页之间的超链接图
- Phase2: PageRankIter
  - 迭代计算各个网页的PageRank值
- Phase3: RankViewer
  - 按PageRank值从大到小输出

Phase1： GraphBuilder

- 原始数据集：维基百科各网页间的链接信息。文本文件，共11.2G。每行包含一个网页名，及其所链接的全部网页名。
- GraphBuilder目标：分析原始数据，建立各个网页之间的链接关系。
  - Map：逐行分析原始数据，输出<URL, (PR_init, link_list)>
    - 其中网页的URL作为key，PageRank初始值(PR_init)和网页的出度列表一起作为value，以字符串表示value，用特定的符号将二者分开。
  - Reduce：输出<URL, (PR_init, link_list)>
    - 该阶段的Reduce不需要做任何处理

Phase2： PageRankIter

- PageRankIter：迭代计算PR值，直到PR值收敛或迭代预定次数。
- Map对上阶段的 <URL, (cur_rank, link_list)>产生两种<key, value>对：
  1. For each *u* in link_list，输出 <*u*, cur_rank/|link_list|>
     - 其中u代表当前URL所链接到网页ID，并作为key；
     - cur_rank为当前URL的PageRank值， |link_list|为当前URL的出度数量，cur_rank/|link_list|作为value。
  2. 同时在迭代过程中，传递每个网页的链接信息<URL, link_list>
     - 在迭代过程中，必须保留网页的局部链出信息，以维护图的结构。

Phase2： PageRankIter

- Reduce 对 Map输出的<URL, link_list> 和多个 <*u*, cur_rank/|link_list|> 做如下处理：
  - 其中<URL, link_list> 为当前URL的链出信息；
  - <*u*, cur_rank/|link_list|>为每个链入网页对当前网页u所贡献的PageRank值，把这些贡献值按公式相加即可得到当前网页u的新的PageRank值。
    - 计算所有val的和，并乘上d，再加上常数(1-d) /N得到new_rank。
    - 输出 (*u*, (new_rank, link_list))。

- 迭代计算公式：
  - $PR(A) = (1-d)/N + d\,(PR(T_1)/C(T_1) + \ldots + PR(T_n)/C(T_n))$

Phase2： PageRankIter


```
1: class Mapper
2:     method Map(nid n, node N)
3:         p ← N.PageRank/|N.AdjacencyList|
4:         Emit(nid n, N)                          ▷ Pass along graph structure
5:         for all nodeid m ∈ N.AdjacencyList do
6:             Emit(nid m, p)                      ▷ Pass PageRank mass to neighbors

1: class Reducer
2:     method Reduce(nid m, [p1, p2, ...])
3:         M ← ∅
4:         for all p ∈ counts [p1, p2, ...] do
5:             if IsNode(p) then
6:                 M ← p                           ▷ Recover graph structure
7:             else
8:                 s ← s + p                       ▷ Sums incoming PageRank contributions
9:         M.PageRank ← s
10:        Emit(nid m, node M)
```

PageRankIter伪代码

Phase3： PageRankViewer

- PageRankViewer：将最终结果排序输出。
  - PageRankViewer从最后一次迭代的结果读出文件，并将文件名和其PR值读出，并以PR值为key，网页名为value，并且以PR值从大到小的顺序输出。
  - 排序过程中可以采用框架自身的排序处理，重载key的比较函数，使其经过shuffle和sort后反序（从大到小）输出。

```
public static class DecFloatWritable extends FloatWritable {
…
@Override
        public int compareTo(Object o) { return  -super.compareTo(o); }
}
```

PageRank迭代终止条件

- 可选的终止条件：
  - 各网页的PageRank值不再改变；
  - 各网页的PageRank值排序不再变化；
  - 迭代至固定次数。

多趟MapReduce的处理


```
public class PageRankDriver {
    private static int times = 10;
    public static void main(String args[]) throws Exception{
            String[] forGB = {"", args[1]+"/Data0"};
            forGB[0] = args[0];
            GraphBuilder.main(forGB);
            String[] forItr = {"Data","Data"};
            for (int i=0; i<times; i++) {
                    forItr[0] = args[1]+"/Data"+(i);
                    forItr[1] = args[1]+"/Data"+(i+1);
                    PageRankIter.main(forItr);
            }
            String[] forRV = {args[1]+"/Data"+times, args[1]+"/FinalRank"};
            PageRankViewer.main(forRV);
    }
}
```

基于MapReduce的图算法小结

- 通常采用邻接表来表示图。
- 通常将一个完整的图结构，分解成若干个局部的子结构，对每个子结构进行并行处理。
- 在map阶段对邻接点产生<key, value>对，经过shuffle和sort后，在reduce阶段更新节点信息。
- 图算法通常是一个迭代过程，上一步的输出作为下一步的输入，由额外的"driver" 控制。

THANK YOU

南京大学
NANJING UNIVERSITY

南京大学计算机软件研究所
Institute of Computer Software, Nanjing University